

# Vectorized Interval Analysis Algorithms and their Applications

Submitted in partial fulfillment of the requirements  
for the degree of

**DOCTOR OF PHILOSOPHY**

by  
**A. K. Prakash**

Guided  
by  
**Prof. P. S. V. Nataraj**

Systems and Control Engineering  
Electrical Engineering Department  
Indian Institute of Technology  
Bombay, India

November 17, 2002

**This thesis is dedicated to  
late Shri. K. Sridhara Menon  
my grandfather**



**ABSTRACT** Although interval analysis algorithms are reliable and accurate, they are often found to execute slowly, especially in difficult problems. The speed limitation has been hitherto overcome by using special coprocessors, functional hardware units, and parallel processing techniques on multi-processor machines. However, as single processor machines such as desktop computers are much more widely used, it would be advantageous to have techniques that speed up interval analysis algorithms on single processor machines.

In this work, we propose *vectorization* as a means of speeding up interval analysis algorithms on single processor machines. Our idea is to process *all* the boxes present in the list at each iteration, using vectorization to perform the various tasks such as function evaluations, bisections, and width checks. Vectorization gives a degree of *parallelism* in scalar processors, enabling faster processing of all boxes in the list at a given iteration. We use the toolbox INTLAB [68] and the Forte FORTRAN 95 compiler [63] for vectorization of interval arithmetic operations. Further, exploiting the feature that *all* the boxes in a list are being processed, we also propose some algorithmic changes wherever possible.

We first apply this new strategy of processing boxes to the interval Newton algorithm [49] for solving a system of nonlinear equations, Neumaier's covering algorithm [61] for solving a parameter - dependent system of nonlinear equations, and the Moore-Skelboe algorithm [51] for solving the unconstrained global optimization problem. We also incorporate some algorithmic changes in the last mentioned algorithm, by modifying the cut-off and monotonicity tests. We show through several test examples that the new strategy considerably speeds up all these algorithms, sometimes as much by a few orders of magnitude.

We then present a set inversion algorithm for characterizing the domain of a set of nonlinear inequalities. The proposed algorithm improves on the one recently proposed by Jaulin *et al.* [31]. The proposed improvements exploit the powerful tool of monotonicity. We test and compare the performances of the proposed and existing algorithms in characterizing the domains of robust stability for two problems, including a case study of speed control of a jet engine. The results of the testing show that the proposed algorithm encloses the domains more accurately than the existing algorithm - meaning that it gives a larger region for which the system stability is guaranteed and a smaller region for which the system stability is indeterminate. The proposed algorithm also requires less space-complexity, but takes more computational time than the existing algorithm.

We next present a derivative free rule for bisection direction selection. The proposed rule is based on the philosophy that "the best bisection direction for the random box of a given list is (likely) the best bisection direction for all other boxes of the list". We evaluate the performance of the proposed rule along with those of several widely used rules of interval analysis, in the context of the template generation problem arising in the QFT approach to robust feedback system synthesis [27]. We conduct these evaluations on a suite of ten real-world problems taken from different engineering application areas. From the tests, we find that the proposed bisection direction selection rule is able to successfully solve *all* examples, whereas *none* of the established rules, such as 'maximum width' and 'maximum smear' rules, are able to do so. Moreover, the proposed bisection direction selection rule gives, on the average, about 58% reduction in the number of solution boxes and 73% reduction in computational time over existing rules. The

findings of the present work also suggest that it may be fruitful to bring in some amount of *randomization* into the body of interval analysis algorithms.

Lastly, we note that there is currently a lack of robust stability analysis methods for polynomials with *nonlinear* parametric dependencies. We therefore propose a vectorized interval analysis algorithm for robust stability analysis of polynomials with nonlinear parametric dependencies. The polynomial coefficients can be any continuous function of the system parameters. The proposed algorithm uses the interval form of the zero exclusion test along with the above proposed derivative free rule for bisection of all boxes in a given list. On a mass-spring-damper system involving thirteen uncertain physical parameters [2], the algorithm is found to successfully verify robust stability in quick time. We believe that the proposed method fills in a significant gap in the robust stability literature.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Vectorization . . . . .	2
1.1.1	How to vectorize? . . . . .	2
1.1.2	Vectorization vs. traditional programming . . . . .	2
1.1.3	Applications of vectorization . . . . .	4
1.1.4	SIMD and vectorization . . . . .	4
1.2	Motivation . . . . .	4
1.3	Main Contributions . . . . .	5
1.3.1	Nonlinear equations . . . . .	5
1.3.2	Parameter - dependent equations . . . . .	6
1.3.3	Global optimization . . . . .	6
1.3.4	Set inversion . . . . .	7
1.3.5	Bisection direction selection . . . . .	7
1.3.6	Robust stability analysis . . . . .	8
1.4	Organization of the thesis . . . . .	8
<b>2</b>	<b>On speeding up the interval Newton algorithm</b>	<b>9</b>
2.1	Introduction . . . . .	9
2.2	Basic algorithm . . . . .	9
2.3	Proposed algorithm . . . . .	11
2.4	Test results . . . . .	12
2.4.1	Performance profiles . . . . .	14
2.5	Conclusions . . . . .	15

<b>3</b>	<b>On speeding up the covering algorithm</b>	<b>25</b>
3.1	Introduction . . . . .	25
3.2	Neumaier's covering algorithm . . . . .	26
3.3	Proposed algorithm . . . . .	27
3.4	Numerical results . . . . .	28
3.5	Conclusions . . . . .	29
<b>4</b>	<b>On speeding up the global optimization algorithm</b>	<b>33</b>
4.1	Introduction . . . . .	33
4.2	Basic algorithm . . . . .	34
4.3	Proposed algorithm . . . . .	35
4.4	Test results . . . . .	36
4.4.1	Termination criterion A . . . . .	38
4.4.2	Termination criterion B . . . . .	39
4.5	Conclusions . . . . .	41
<b>5</b>	<b>An improved algorithm for set inversion</b>	<b>69</b>
5.1	Introduction . . . . .	69
5.2	Basic algorithm . . . . .	70
5.3	Proposed Algorithm . . . . .	70
5.3.1	Throwing parts of box (TPB) . . . . .	70
5.3.2	The algorithm . . . . .	72
5.4	Test example: Polynomial stability . . . . .	72
5.5	Case study: Speed control of jet engine . . . . .	73
5.5.1	Vertex-type procedure . . . . .	73
5.5.2	Obtaining compensator parameters . . . . .	74
5.6	Conclusions . . . . .	76
<b>6</b>	<b>A new rule for bisection direction selection</b>	<b>81</b>
6.1	Introduction . . . . .	81
6.2	Bisection Direction Selection Rules . . . . .	82
6.2.1	Rule A (Maximum width) . . . . .	82
6.2.2	Rule B (Scaled maximum width) . . . . .	83
6.2.3	Rule C (Maximum smear) . . . . .	83
6.3	A new rule for bisection direction selection . . . . .	83
6.4	Application: template generation . . . . .	84
6.4.1	Ranking . . . . .	85
6.4.2	Averaging . . . . .	86
6.4.3	Performance profiling . . . . .	86

6.5	Summary . . . . .	86
<b>7</b>	<b>An algorithm for robust stability analysis</b>	<b>93</b>
7.1	Introduction . . . . .	93
7.2	Background . . . . .	94
7.3	Proposed algorithm . . . . .	94
7.4	Application: mass-spring-damper . . . . .	95
7.5	Conclusions . . . . .	96
<b>8</b>	<b>Conclusions</b>	<b>99</b>
8.1	Suggestions for future work . . . . .	102
<b>A</b>	<b>Test problems used in chapter 2</b>	<b>103</b>
<b>B</b>	<b>Test problems used in chapter 3</b>	<b>109</b>
<b>C</b>	<b>Test problems used in chapter 6</b>	<b>111</b>
<b>D</b>	<b>Interval analysis</b>	<b>115</b>
	<b>References</b>	<b>117</b>





# List of Figures

2.1	Performance profiles of time and maximum list length for $\varepsilon = 10^{-4}$ . . . . .	16
2.2	Performance profiles of time and maximum list length for $\varepsilon = 10^{-6}$ . . . . .	16
4.1	Performance profile plots for number of function evaluations with termination criterion A. . . . .	43
4.2	Performance profile plots for <i>flops</i> with termination criterion A. . . . .	43
4.3	Performance profile plots for computational time (seconds) with termination criterion A. . . . .	44
4.4	Performance profile plots for maximum list length with termination criterion A. . . . .	44
4.5	Performance profile plots for number function evaluations with termination criterion B. . . . .	45
4.6	Performance profile plots for <i>flops</i> with termination criterion B. . . . .	45
4.7	Performance profile plots for computational time with termination criterion B. . . . .	46
4.8	Performance profile plots for maximum list length with termination criterion B. . . . .	46
5.1	Plot of the domain of guaranteed robust stability for the polynomial example. . . . .	77
5.2	Plot of the domain of indeterminate robust stability for the polynomial example. . . . .	77
5.3	Block diagram of compressor speed control loop of jet engine. . . . .	78
5.4	Plot of the domain of guaranteed robust stability for the speed control loop of jet engine. . . . .	78
5.5	Plot of the domain of indeterminate robust stability for the speed control loop of jet engine. . . . .	79
5.6	Responses of the compressor speed control system to a unit step input. . . . .	79

6.1	Performance profile plots for the number of solution boxes and computational time metrics. . . . .	88
7.1	Schematic representation of the mass-spring-damper system. . . . .	97

# List of Tables

2.1	Performance comparison of algorithms for $\epsilon = 10^{-4}$ . . . . .	17
2.2	Performance comparison of algorithms for $\epsilon = 10^{-6}$ . . . . .	21
3.1	Comparisons of algorithms for $\epsilon = 0.01$ . . . . .	30
3.2	Comparisons of algorithms for $\epsilon = 0.001$ . . . . .	31
4.1	Rankings with termination criterion A. . . . .	47
4.2	Statistical measures with termination criterion A. . . . .	47
4.3	Minimum, Mean, and Maximum of Ratios and Percent Reductions with termination criterion A. . . . .	47
4.4	Rankings with termination criterion B. . . . .	48
4.5	Statistical measures with termination criterion B. . . . .	48
4.6	Minimum, Mean, and Maximum of Ratios and Percent Reductions with termination criterion B. . . . .	48
4.7	Function evaluations for basic and proposed algorithms with termination criterion A. . . . .	49
4.8	<i>Flops</i> for basic and proposed algorithms with termination criterion A. . . . .	51
4.9	Computational time for basic and proposed algorithms with termination criterion A. . . . .	53
4.10	Maximum list length for basic and proposed algorithms with termination criterion A. . . . .	55
4.11	Function evaluations for basic and proposed algorithms with termination criterion B. . . . .	57
4.12	<i>Flops</i> for basic and proposed algorithms with termination criterion B. . . . .	59

4.13	Computational time for basic and proposed algorithms with termination criterion B. . . . .	61
4.14	Maximum list length for basic and proposed algorithms with termination criterion B. . . . .	63
4.15	Domains and computed global minimums and minimizers with termination criterion A. . . . .	65
5.1	Performance comparison of set inversion algorithms for characterization of the domain of robust stability for a test polynomial. . . . .	80
5.2	Performance comparison of set inversion algorithms for the jet engine control problem. . . . .	80
6.1	Performance of the proposed bisection rule over different runs. . . . .	89
6.2	Comparison of the 'best bisection direction' $k$ for the random box versus the actual 'best bisection direction' found for all other boxes in the list over various iterations . . . . .	90
6.3	Number of solution boxes obtained with different bisection rules. . . . .	90
6.4	Computational time taken with different bisection rules. . . . .	91
6.5	Ranking of rules based on the number of solution boxes. . . . .	91
6.6	Ranking of rules based on the computational time taken. . . . .	91
6.7	Percentage reduction obtained with the proposed rule over existing rules. . .	92

## List of Symbols

$\mathbf{A}$	– an interval matrix
$b$	– number of solution intervals (boxes)
$C$	– complex plane
$\mathbf{C}$	– a subbox for which $f > 0$ is infeasible
$d$	– a matrix
$\mathbf{D}$	– an interval matrix in $\Re^n$
$D_i(\mathbf{X})$	– merit function
$f$	– real function
$f'$	– Jacobian of $f$
$\bar{f}$	– the exact range of $f$
$fe$	– number of function evaluations
$F'$	– inclusion function for $f'$
$F$	– inclusion function for $f$
$F_{ang}$	– inclusion function for phase angle
$F_{mag}$	– inclusion function for magnitude
$ge$	– number of gradient evaluations
$G(\mathbf{X})$	– derivative of $F$
$i, j$	– indexes
$J$	– dimension of the merit function
$k$	– a coordinate direction for bisection
$l_r$	– the length of list $L$ at a given iteration
$l_r'$	– the (temporary)length of list $L$ at a given iteration
$L$	– list
$L_{sol}$	– solution list
$m(\mathbf{X})$	– midpoint of $\mathbf{X}$
$l, m, n$	– dimensions
$n_p$	– number of test functions
$n_s$	– number of algorithms (solvers)
$ml$	– maximum list length
$\mathcal{M}$	– solution set
$p$	– a test function
$\mathcal{P}$	– test set of functions
$q$	– parameter vector
$\mathbf{Q}^0$	– bounding box for $q$

$r$	– rank
$r_{p,s}$	– performance ratio
$R$	– preconditioning matrix
$\Re$	– set of all real numbers
$s$	– Laplace variable
$t$	– time in seconds
$t_{p,s}$	– computing time required to solve a test function $p$ by an algorithm
$v^1, v^2$	– minimum value for $\mathbf{V}^1$ and $\mathbf{V}^2$
$\mathbf{V}^1, \mathbf{V}^2$	– the subboxes obtained by bisection
$w(\mathbf{X})$	– width of an interval $\mathbf{X}$
$\mathbf{X}$	– an interval (box)
$\mathbf{X}^0$	– initial box
$\mathbf{X}^*$	– random box from the list
$z$	– the second component of any pair in list $L$
$\mathbf{Z}$	– the first component of any pair in list $L$
$\alpha$	– number of iterations for which $> 90\%$ of the boxes in the list follow a particular direction for bisection
$\mathcal{D}$	– a subset
$\varepsilon$	– desired accuracy
$\Lambda$	– a box of system parameters
$\lambda$	– system parameter vector
$\mathcal{G}$	– template of a system
$\rho_s(\tau)$	– Performance profile

# 1

## Introduction

*"The interval concept is on the  
 border line, linking pure mathematics  
 with reality and pure analysis with  
 applied analysis."*

*T. Sunaga*

Interval analysis enables the design of algorithms that do the approximation and a rigorous error analysis in a single computation. With interval approaches, one can directly deal with interval sets containing finitely many points, and perform set operations such as unions, intersections, subdivisions, find convex hulls, test for set inclusion, etc.

First introduced by Wamus [80] and Sunaga [74], the turning point in the history of interval analysis came with the appearance of Moore's book [48]. Since then, several books and papers describing the evolution of the area have appeared, see, for instance, [4], [25], [34], [49], [62], and the references cited therein. The tools of interval analysis have been widely applied in diverse areas of engineering and sciences such as control systems, electrical circuits, finance, structural analysis, and robotics. An overview of the applications is given in [31], [36], and [40].

On the other hand, the increasing availability of processors with advanced architecture has had a significant effect on all spheres of scientific computation, including algorithm research and software development. Modern complex processors with multiple execution units, super scalar pipelined architectures, very long instruction word, and vector processing units, etc., challenge one to take full advantage of the processing capabilities. Today, general-purpose processors are so fast that the performance of many applications is limited by the memory



bandwidth, and not by the processor speed. Advanced cache architectures are an attempt to overcome this limitation, and proper ‘vectorization’ can be the key to a cost-effective solution. Vectorization considerably reduces the frequency with which data are moved between the different levels of the memory hierarchy in order to attain high performance. Consequently, better management of memory and better cache utilization are obtained through vectorization.

A drawback with existing basic interval analysis algorithms is that although they are reliable and accurate, they usually execute slowly for ‘difficult’ problems. The work in this thesis primarily advances vectorization as a means for speeding up interval analysis algorithms.

## 1.1 Vectorization

Vectorization, or vectorized programming, uses a single statement to apply an operation to every element of a list or an array, in contrast to conventional scalar code that is a *one-at-a-time* process. In a scalar code, each element must finish the final step of processing before the next element can begin the first step, that is, each loop iteration begins when the previous iteration ends. With vectorized code, each element begins the first step when the previous element finishes the first step rather than the final step. Thus, vectorization is the process by which many common algorithms can be optimized to maximize processor utilization and memory bandwidth, yielding much more efficient implementations of the algorithms.

### 1.1.1 How to vectorize?

The most commonly used approach is to write a *vector* library. The library functions may be written in a high-level language such as C or FORTRAN, or even in assembly language for a particular processor. The application programmer then uses vectorization in his or her application by calling these vector library routines. Such types of vector libraries are available, for instance, with MATLAB [47], FORTRAN [63], and PASCAL / Delphi <sup>1</sup>.

### 1.1.2 Vectorization vs. traditional programming

The DRAM in a computing machine has two types of sequences to access its contents <sup>2</sup>:

1. If the current access is in the same page as the previous access, then the memory location can be selected very rapidly, and the data accessed potentially without wait states.

---

<sup>1</sup><http://www.optivec.com>

<sup>2</sup><http://www.skycomputers.com/technical/vectorization.html>

2. If the current access requires some page change, then the page must be selected before the location of the page. Changing pages like this takes several clocks, and induces wait states. The time consumed by the wait states can be twice the time used to actually transfer the data. This wait process is repeated until the entire computation is performed. Such wait states are unavoidable in normal scalar kind of programming on the usual desktop computers. The result is that an enormous amount of time is needed to deal with similar type of elements, one by one, in a long list or array.

The wait states accessing the data from memory must be eliminated if the performance of a calculation is to be improved. This is done by changing the order in which the operations are performed. If the processor were to pre-fetch all of the data in a vector at once, loading it into the L1 cache, then the memory accesses would tend to be in the same page. With a good memory controller and a good processor, these memory accesses can then be performed with few, and perhaps no, wait states. The data will stream into the processor at the maximum bandwidth possible from the memory.

The overall result is a function that performs faster than the original scalar code, as long as the source and destination vectors all fit into the L1 cache simultaneously. If they do not fit into the cache, then the cache will start thrashing. For example, if each individual vector is larger than the cache, loading the vector will discard the previous vector from cache. By the time the computation is performed, the behavior of the processor will revert back to the scalar case, and all the work involved in attempting to pre-fetch the data ends up being an overhead without any benefit. An additional optimization is possible to deal with this issue.

If the source and destination vectors do not fit into cache, the problem is broken down into pieces so that the pieces do fit into cache, if the maximum performance is to be achieved. This technique is called “strip mining”. The process involved is to pre-fetch a portion of the source vectors, and make room for a portion of the destination vector. The computation is performed, and the result is flushed to memory. Then, the next portions of the input and output vectors are set up in cache, the computation performed, and the result flushed to memory. This continues until the entire computation is performed. The vector portions are called “strips”, hence the name “strip mining”. Ideally, the size of the strips is computed so that they just fit into the L1 cache. If the strip size is too large, the strips will overflow the cache, inducing memory bandwidth overhead. If the strips are too small, additional overhead is induced from managing the smaller strips. Further, the pointers into the vectors must be managed correctly to produce the correct result with minimum overhead.

There are additional optimizations that are considered when performing a sequence or a chain of vector computations. If intermediate results can stay in the cache, they do not take any memory bandwidth at all when they are used later in computations. In addition, if a vector can somehow get marked as temporary when it will not be used elsewhere in the program, then it is possible to avoid using memory bandwidth to store it back into memory

by invalidating those cache lines. Strip mining and careful cache manipulation can make dramatic improvements in application performance.

### 1.1.3 Applications of vectorization

Vectorization has applications where similar data held in a list or an array are to be processed. Some of these applications are in digital signal processing [43], NMR data processing [23], calculation of molecular properties based on quantum mechanics [21], numerical linear algebra [18], and load flow analysis [29]. These applications are characterized by having their data stored in contiguous memory locations as vectors or arrays, and can be optimized through the use of vectorization to achieve additional performance gains. Another potential use of vectorization is in the area of interval analysis, where intervals are stored in a list.

### 1.1.4 SIMD and vectorization

Recently developed processors include ‘Single Instruction, Multiple Data’ (SIMD) commands that allow computers to do several mathematical computations simultaneously [9], [83]. For instance, Motorola chips used in the Macintosh series have AltiVec instructions, INTEL chips support SSE and SSE2 instructions, while AMD chips support 3DNow! instructions. Programs that support these instructions can potentially run much more quickly. Some compilers, such as Vector C<sup>3</sup>, are able to tap the potential of SIMD features available with the latest series of microprocessors. The high-performance INTEL C++/FORTRAN compiler is one of the first commercially available vectorizing compilers that targets the streaming-SIMD-extensions (SSE/SSE2). A detailed overview of the vectorization techniques used by this compiler can be found in [10]. This kind of automatic exploitation of SIMD facilities in single processor machine using vectorization leads to a definite improvement in the speed and efficiency of scientific computations.

## 1.2 Motivation

Over the last decade or so, several software packages and libraries have been developed to perform interval arithmetic and improve the accuracy and reliability of numerical computations, such as INTLIB [35], INTBIS [37], INTLIB [35], PASCAL-XSC [38], C-XSC [42], PROFIL/BLAS [39] and INTLAB[68]. The main disadvantage of these self validating interval analysis software is their *speed*. Since arithmetic operations are simulated in software, a tremendous amount of overhead is incurred for function calls, memory management, error and range checking, expression manipulation, changing rounding modes, exception handling,

---

<sup>3</sup><http://www.codeplay.com>

etc. The overhead leads to an implementation of interval arithmetic that is considerably slow to execute on a computer architecture designed to handle only floating point numbers.

The speed limitation can be somewhat overcome by using special purpose coprocessors [70] or functional units [22] to deal with interval arithmetic. Parallel processing techniques on multi-processor machines can also be exploited to speed up interval methods [20], [28], [45], [46], [69] and [72].

However, as single processor based computing machines such as desktop computers are more widely used, it would be advantageous to have techniques that speed up the algorithm on *single* processor machines. In this thesis, the focus is on applying vectorization to speed up some key interval analysis algorithms implemented on normal desktop computers with single processors.

### 1.3 Main Contributions

All basic interval algorithms involve maintenance of a list of boxes to be processed. At each iteration of a typical interval analysis algorithm, boxes are picked up from the list, one at a time, and processed. The iterations are continued till the list becomes empty or termination criteria is satisfied.

In this work, vectorization is proposed as a means of speeding up some basic interval algorithms. The idea is to process all the boxes present in the list simultaneously at each iteration, and use vectorization to perform the various tasks such as function evaluations, bisection, width checking, etc. The toolbox INTLAB [68] and the FORTE FORTRAN 95 compiler [63] are used for vectorization of interval arithmetic operations. Further, exploiting the feature that all the boxes in a list are being processed, some algorithmic changes are also made wherever possible.

#### 1.3.1 Nonlinear equations

The interval Newton algorithm [48] reliably computes to a prescribed accuracy *all* solutions of a nonlinear system of equations  $f(x) = 0$  for a continuously differentiable function  $f : \mathbb{R}^l \rightarrow \mathbb{R}^l$  in a given interval vector (or box)  $\mathbf{X}^0$ . In this work, we present a vectorized version of the interval Newton algorithm. As is well known, in each iteration of the interval Newton algorithm, we choose only one box from the list for processing. In each iteration of the proposed algorithm, we choose *all* boxes from the list for processing. By doing so, we do not alter the essence of the interval Newton algorithm, because in any case all the boxes in the list have to be processed sooner or later. However, because no box is kept waiting in the list for its turn (which may come after many iterations), and instead every box is processed right away in the very same iteration, we expect our strategy to result in a considerable speed-up of the algorithm. To perform function and gradient evaluations, zero exclusion checks, Gauss

- Seidel steps, width checks, and bisections on *all* boxes in an iteration, we use *vectorized* interval arithmetic operations. We test and compare the performances of the proposed and original algorithms on a set of twenty-one test problems. The test results show the proposed algorithm to be considerably faster (on the average, more than eleven times faster) and to require less memory and computational effort than the original interval Newton algorithm in all problems.

### 1.3.2 Parameter - dependent equations

Neumaier [61] proposed the so-called covering algorithm for enclosing the solution set of parameter - dependent systems of nonlinear equations. However, in Neumaier's covering algorithm only one box is processed in each iteration. Such processing is inherently slow, due to its *sequential* nature. On the other hand, in each iteration of the covering algorithm it is clearly possible to simultaneously process *all* boxes that are present (this can be done without altering in any way the essence of the algorithm). The strategy results in greatly speeding up the algorithm, because all boxes present in every iteration are processed simultaneously. We call this version of the covering algorithm as the *vectorized* covering algorithm. This work presents a vectorized version of the covering algorithm, in which *all* boxes present are processed *simultaneously* in each iteration. It is shown through several examples that the vectorized algorithm is significantly faster (by up to 2 orders of magnitude in demanding problems) than the original algorithm.

### 1.3.3 Global optimization

A basic branch and bound algorithm of interval analysis to solve the unconstrained global optimization is the Moore-Skelboe algorithm [51], augmented with the midpoint test of Ichida and Fujii [30] and the monotonicity test [65]. Although this basic algorithm is reliable, it is sometimes slow for 'difficult' problems. In this work, we propose some modifications to speed up the basic interval global optimization algorithm, as follows.

- In the basic algorithm we choose for processing *only* the first box from the list at each iteration. In the proposed algorithm, we choose and process *all* boxes from the list at each iteration.
- Using the fact that we are processing all the boxes simultaneously, we propose to use a new cut-off test (also called as the midpoint test) in the algorithm. In the basic algorithm, the cut-off value is the function value at the midpoint of the *leading* box at each iteration. In the proposed algorithm, the cut-off value is the minimum of the function values at the midpoints of *all* boxes in the list at each iteration. Hence, the new cut-off test is expected to discard more irrelevant boxes at each iteration.

- In the basic algorithm, the monotonicity test is applied to the subboxes obtained by bisection of the *leading* box at each iteration. In the proposed algorithm, the monotonicity test is applied to the subboxes obtained by bisection of *all* boxes in the list at each iteration.
- To perform function and gradient evaluations, monotonicity test, cut-off test, width checks, and bisections on *all* boxes in the list at an iteration, we propose to use *vectorized* interval arithmetic operations.

Test results on a collection of fifty standard test functions with two different termination criteria show that the proposed algorithm uses any extra function evaluations, computational effort, and maximum list length *effectively*, yielding considerable savings in computational time.

#### 1.3.4 Set inversion

We present an algorithm to characterize the set  $\mathcal{S} = \{x \in \mathbb{R}^l : f(x) > 0\} = f^{-1}([0, \infty[^m)$  in the frame work of set inversion using interval analysis. The proposed algorithm improves on the algorithm of Jaulin *et al.* [31]. The improvements exploit the powerful tool of monotonicity. We test and compare the performance of the proposed and existing algorithms in characterizing the domains of robust stability for two problems, including a case study of speed control of a jet engine. The results of the testing indicate that the proposed algorithm encloses  $\mathcal{S}$  more accurately than the existing one - meaning that it gives a larger region for which the system stability is guaranteed and a smaller region for which the system stability is indeterminate. The proposed algorithm also requires less space-complexity, but takes more computational time than the existing algorithm.

#### 1.3.5 Bisection direction selection

A fundamental problem in numerical analysis is that of computing the range of values of a function  $f$  on a box  $\mathbf{X}$ , denoted as  $\bar{f}(\mathbf{X})$ . Since in general it is not possible to compute the exact range  $\bar{f}(\mathbf{X})$ , we therefore consider the problem of finding an interval enclosure of  $\bar{f}(\mathbf{X})$  with a desired degree of accuracy  $\varepsilon$ . A recently proposed interval analysis based algorithm for range computations, shown to be usually more efficient than other similar algorithms, is the one of Nataraj and Sheela [58].

In this work, we propose a new derivative free rule for bisection direction selection in the above interval analysis algorithm for range computations. In the proposed rule, at each iteration of the algorithm, we pick a box *randomly* from the current list and actually find out which is the ‘best bisection direction’ for the random box. We then take this ‘best bisection direction’ for the random box as also the ‘best bisection direction’ for all other boxes of current list, and bisect all these boxes in the said direction. The philosophy behind the proposed rule

is that “the best bisection direction for the random box of current list is likely the best bisection direction for all other boxes of current list”. The performance of the proposed rule is then compared with those of several widely used bisection direction selection rules of interval analysis on a suite of ten real-world problems, taken from different engineering application areas. The performance evaluation is done in the context of template generation problem arising in the QFT approach to robust feedback system synthesis [27]. It is found that, on the average, about 58% reduction in the number of solution boxes and 73% reduction in computational time is obtained with the proposed bisection direction selection rule over existing rules.

### 1.3.6 Robust stability analysis

We present an interval analysis algorithm for robust stability analysis of polynomials with nonlinear parametric dependencies. The proposed algorithm is based on an interval zero exclusion test, the random bisection strategy, and vectorization technique. The proposed algorithm is applied to a mass-spring-damper system with a large number of uncertain parameters, and found to verify robust stability in quick time.

## 1.4 Organization of the thesis

The rest of the thesis is organized as follows. In chapter 2, we present a vectorized version of the interval Newton algorithm for solving systems of nonlinear equations. We then test and compare the performances of the proposed and original algorithms on a set of twenty-one test problems drawn from various fields. In chapter 3, we propose a vectorized version of the covering algorithm for solving parameter - dependent systems of nonlinear equations. We show the superiority of the proposed algorithm through several examples. In chapter 4, we propose a vectorized version of the basic Moore - Skelboe algorithm for solving the unconstrained nonlinear global optimization problem. We evaluate its performance on a collection of fifty standard test functions with two different termination criteria. In chapter 5, we present an improved algorithm for set inversion via interval analysis and demonstrate its capabilities for speed control of a jet engine. In chapter 6, we propose a new rule for bisection direction selection in the context of range finding algorithms, and apply it to generate templates or value sets for ten real-world systems. In chapter 7, we present an algorithm for robust stability analysis of control systems and demonstrate its capability on a 13 parameter mass -spring - damper control system. In chapter 8, we give the conclusions of the present work and some suggestions for future work. In appendix A, B, and C we give test problems used in chapters 1, 2 and 6 respectively. Appendix D gives notations and definitions of some of the terms of interval analysis used in this thesis.

## 2

# On speeding up the interval Newton algorithm

## 2.1 Introduction

The problem of finding the solution vectors of a system of nonlinear equations is addressed in this chapter. The interval Newton algorithm [48] finds *all* solutions of a nonlinear system of equations  $f(x) = 0$  for a continuously differentiable function  $f : \mathbb{R}^l \rightarrow \mathbb{R}^l$  in a given interval vector (or box)  $\mathbf{X}^0 \subseteq \mathbb{R}^l$ .

In this chapter, we propose a modification that considerably speeds-up the interval Newton algorithm. As is well known, in each iteration of the interval Newton algorithm, we choose only one box from the list for processing. In each iteration of the proposed algorithm, we choose *all* boxes from the list and use vectorization for processing. This is the only difference between the proposed and original algorithms, and clearly, it does not alter the essence of the algorithm.

We also test and compare the performances of the proposed and original algorithms on twenty-one problems, and show the proposed algorithm to be considerably faster (on the average more than eleven times faster) than the original one in all examples.

## 2.2 Basic algorithm

There are several variants and embellishments of the interval Newton algorithm, see, for instance, [4], [24], [26], [34], [62]. We consider a model interval Newton algorithm that has the *basic* features.

### **A Model Interval Newton Algorithm**

Inputs: the initial box  $\mathbf{X}^0$ , inclusion function of the given function and its Jacobian (denoted respectively as  $F$  and  $F'$ ), and a parameter  $\varepsilon$  to check if the width of a box is small.



Begin Algorithm

1. (Initialization step) : Initialize list  $L = \{\mathbf{X}^0\}$ .
2. (Start new iteration with a box from  $L$ ) : If the list  $L$  is empty, go to step 8. Else, choose a box from the list  $L$ , denote it as current box  $\mathbf{X}$ , and remove  $\mathbf{X}$  from  $L$ .
3. (Do zero exclusion test on current box) : Evaluate the inclusion function  $F$  on the box  $\mathbf{X}$ . If  $0 \notin F(\mathbf{X})$ , discard  $\mathbf{X}$ , and go to step 2.
4. (Set up a linear interval equation for current box, and solve using the interval Gauss-Seidel method) :
  - (a) Construct the Jacobian matrix  $F'(\mathbf{X})$ , and the inverse midpoint preconditioning matrix  $R = \{m(F'(\mathbf{X}))\}^{-1}$ , where  $m(\cdot)$  denotes the midpoint.
  - (b) Find the midpoint of the box  $\mathbf{X}$  as  $\tilde{x} = m(\mathbf{X})$ , and evaluate  $f$  at  $\tilde{x}$ .
  - (c) Using the interval Gauss-Seidel (IGS) method [34], [62], solve the linear interval equation
 
$$RF'(\mathbf{X})(\mathbf{X} - \tilde{x}) = -Rf(\tilde{x})$$

The solution obtained may be empty, a single box, or several boxes (as when gaps arise in one or more solution components).
  - (d) If the solution is empty, discard  $\mathbf{X}$ . If the solution is a single box, then add the midpoint  $\tilde{x}$ , and replace  $\mathbf{X}$  with the resulting box as current box. If there are several solution boxes, do likewise for each, and denote them as current boxes.
5. (Check box width and print solution) : For (each) current box  $\mathbf{X}$ , do the following. Find the width of  $\mathbf{X}$ ; if the width does not exceed  $\varepsilon$ , print out  $\mathbf{X}$  as a solution box, and discard it. If no boxes remain, return to step 2.
6. (Bisection) : For (each) current box  $\mathbf{X}$ , bisect box along the maximum width coordinate direction, and enter the resulting subboxes into the list  $L$ .
7. (Terminate current iteration) : Go to step 2.
8. EXIT algorithm.

End Algorithm.

**Remark 2.1** *In steps 3 and 4, we discard irrelevant parts of the box using the zero exclusion test and the interval Gauss-Seidel method.*

## 2.3 Proposed algorithm

In the model interval Newton algorithm, in each iteration we choose only one box from the list  $L$  for processing. On the other hand, in each iteration we can choose *all* boxes from the list for processing. By doing so, we do not alter the essence of the interval Newton algorithm, because in any case all the boxes in the list have to be processed sooner or later. However, because no box is kept waiting in the list for its turn (which may come after many iterations), and instead every box is processed right away in the very same iteration, we expect our strategy to result in a considerable speed-up of the algorithm.

To perform function and gradient evaluations, zero exclusion checks, Gauss-Seidel steps, width checks, and bisections on *all* boxes in an iteration, we use *vectorized* interval arithmetic operations. We could have also used FOR loops that run over all the boxes to do the same things, but we follow [47] which strongly advocates employment of vectorization instead of FOR loops wherever possible.

In the following example, we illustrate how vectorized function evaluation can be done on all boxes from a list.

**Example 2.1** *Consider, for example, the 1- dimensional function in [61]*

$$f(x_1, x_2) = x_1^3 - x_1x_2^2 + x_1^2 - x_1x_2 - x_2^2$$

*and suppose we want to evaluate  $F$  over all boxes from a list. Then, using the notation of INTLAB (which is based on MATLAB), we can do this using the single program statement*

$$F = \text{power}(\mathbf{X}(:, 1), 3) - \mathbf{X}(:, 1) .* \text{sqr}(\mathbf{X}(:, 2)) + \text{sqr}(\mathbf{X}(:, 1)) - \mathbf{X}(:, 1) .* \mathbf{X}(:, 2) - \text{sqr}(\mathbf{X}(:, 2))$$

*where,  $\mathbf{X}(:, 1)$  denotes  $\mathbf{X}_1$  for all boxes and  $\mathbf{X}(:, 2)$  denotes  $\mathbf{X}_2$  for all boxes. The function evaluation over all the boxes is done with this statement, because the operations  $+$ ,  $-$ ,  $.*$ ,  $\text{sqr}$ , and  $\text{power}$  are performed element-wise between vectors (cf. [47]) and INTLAB overloads these ordinary arithmetic operations with the corresponding vectorized interval arithmetic operations [68].*

In a similar way, we can perform gradient evaluations, width checks, etc., on all boxes from a list.

We next present our algorithm that is based on the above idea.

### A vectorized Interval Newton Algorithm

Begin Algorithm

1. (Initialization step) : Initialize list  $L = \{\mathbf{X}^0\}$ .
2. (Start new iteration with all boxes from  $L$ ) : If the list  $L$  is empty, go to step 8. Else, choose all boxes from the list  $L$ , and remove them from  $L$ .

3. (Do the zero exclusion test on all boxes) : Evaluate the inclusion function  $F$  on all boxes, and discard all those boxes  $\mathbf{X}$  for which  $0 \notin F(\mathbf{X})$ . If no more boxes remain, go to step 8.
4. (Set up linear interval equations for all boxes, and solve using the interval Gauss-Seidel method) :
  - (a) Construct the Jacobian matrices  $F'(\mathbf{X})$  and the inverse midpoint preconditioning  $R = \{m(F'(\mathbf{X}))\}^{-1}$ , for all boxes.
  - (b) Find the midpoint  $\tilde{x} = m(\mathbf{X})$  and evaluate  $f(\tilde{x})$ , for all boxes.
  - (c) Apply the interval Gauss-Seidel (IGS) method to solve the corresponding linear interval equation
 
$$RF'(\mathbf{X})(\mathbf{X} - \tilde{x}) = -Rf(\tilde{x})$$
 for all boxes.
  - (d) Discard all those boxes for which the corresponding linear interval systems are found incompatible (i.e., solution set is empty). If no more boxes remain, go to step 8. Else, add the respective midpoint  $\tilde{x}$  to each solution box, and proceed with these as the current boxes.
5. (Check widths of all boxes and print solutions) : Find the widths of all boxes. Print all those boxes whose widths do not exceed  $\varepsilon$  and discard them. If no more boxes remain, go to step 8.
6. (Bisect all boxes into two subboxes each) : Find the maximum width coordinate directions for all boxes, and bisect the boxes along the respective directions. Enter the resulting subboxes into the list  $L$ .
7. (Terminate current iteration) : Go to step 2.
8. EXIT algorithm.

END Algorithm.

## 2.4 Test results

We consider some test problems for comparing the performances of the proposed and model interval Newton algorithms. The test problems considered are listed in the Appendix A. We carry out all computations on a single processor PC/Pentium-III 800 MHz machine with 256 MB RAM using version 3 of INTLAB [68].

Tables 2.1 and 2.2 give the results for the various test problems for two values of  $\varepsilon$ , that is,  $\varepsilon = 10^{-4}$  and  $10^{-6}$ . We use the following efficiency measures (the notation used for the entries in the tables is given in brackets) :

- Computational time in seconds (t)
- Number of solution boxes (b)
- Maximum list size (ml)
- Number of functional evaluations (fe)
- Number of gradient evaluations (ge)
- Number of floating operations (*flops*)

From the results given in the Tables, we make the following observations:

1. In all examples, the number of solution boxes obtained is the same with either algorithm.
2. In all examples, the number of function evaluations is the same in either algorithm.
3. In all examples, the number of gradient evaluations is the same in either algorithm.
4. In all examples, the maximum list size is slightly less in the proposed algorithm.
5. In all examples, the computational effort (in terms of *flops*) is slightly less in the proposed algorithm. This can be attributed to the fact that in INTLAB, interval arithmetic and slopes are done by operator overloading, which incurs substantial overhead. This overhead is needed once for each box in the model algorithm, but only once per iteration in the vectorized version [60].
6. In all examples, the vectorized algorithm is considerably faster than the model algorithm.
  - Regarding the speed-up factor obtained with the proposed algorithm relative to the model algorithm : The minimum speed-up is 1.5 and occurs for problem Toolbox2. The maximum speed-up is 22.8 and occurs for problem Neuro6. The average speed-up over all examples is about 11.2 with either accuracy .
  - Regarding the percentage reduction in computational time obtained with the vectorized algorithm relative to the model algorithm : The minimum reduction is 31.8% and occurs for problem Toolbox2. The maximum reduction is 95.6% and occurs for problem Neuro6. The average reduction over all examples is about 84.7% with either accuracy.

- A reduction in time above 85% is obtained in most problems.

**Remark 2.2** *We note that the vectorized algorithm achieves significant speed-ups even on ordinary desktop PCs (as in our tests). There is no requirement for any parallel or vector processors.*

**Remark 2.3** *The actual speed-up factor and reduction in computational time perhaps vary considerably with the computing environment used, and may be less conspicuous in programming languages where control structures are more efficiently implemented than in MATLAB.*

#### 2.4.1 Performance profiles

Performance profile is proposed as a tool for evaluating and comparing performance of algorithms in [17]. The performance profile for an algorithm is the (cumulative) distribution function for a performance metric. Performance profiles eliminate the influence of a small number of problems on the final evaluation conclusions.

For computational time as the performance metric, performance profiles can be generated as follows. Let  $\mathcal{P}$  be the test set of functions,  $n_s$  be the number of algorithms and  $n_p$  be the number of functions. For each test function  $p$  and algorithm  $s$ , define

$$t_{p,s} = \text{computing time required to solve a test function } p \text{ by algorithm } s$$

The performance ratio for computation time is calculated as

$$r_{p,s} = \frac{t_{p,s}}{\min \{t_{p,s} : 1 \leq s \leq n_s\}}$$

We choose a parameter  $r_M \geq r_{p,s}$  for all  $p, s$ , such that  $r_{p,s} = r_M$  if and only if algorithm does not solve the test function  $p$ . Now, the performance profile for computing time can be defined as

$$\rho_s(\tau) = \frac{1}{n_p} \text{size} \{p \in \mathcal{P} : r_{p,s} \leq \tau\}$$

Similarly, performance profiles for any performance metric can be defined.

The following observations are made from the performance profile plots computed for time and maximum list length efficiency measures (see Figures 2.1 and 2.2).

Computational time and maximum list length.

Performance profile plots for computational time in Figures 2.1 and 2.2 show that the proposed algorithm is able to solve all the problems for  $\tau = 1$ , and the basic algorithm solves all the problems for  $\tau < 25$ , using both the accuracies. Performance profile plots for maximum list length Figures 2.1 and 2.2 show that both the algorithm require almost same maximum list lengths.

## 2.5 Conclusions

A vectorized interval Newton algorithm was proposed for solving finite-dimensional systems of nonlinear equations. It was demonstrated through several test examples that the proposed algorithm was considerably faster, on the average, more than eleven times faster than the model algorithm.

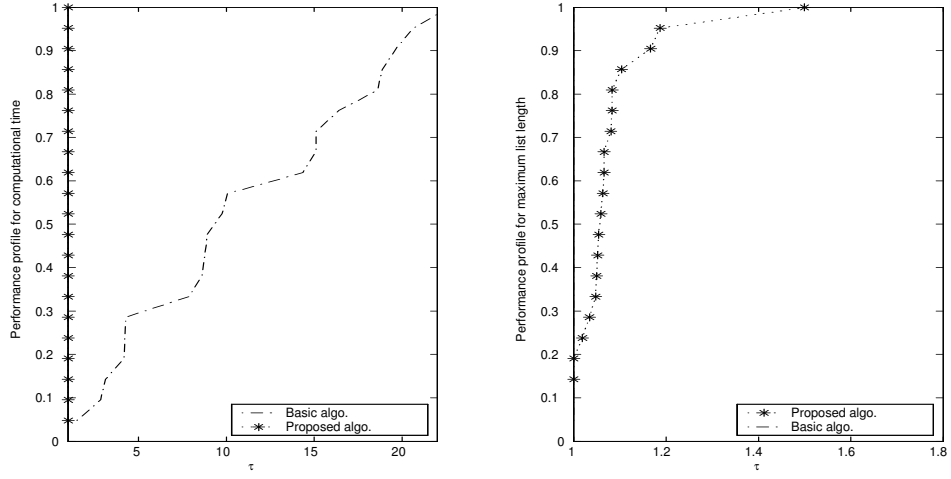


FIGURE 2.1. Performance profiles of time and maximum list length for  $\varepsilon = 10^{-4}$ .

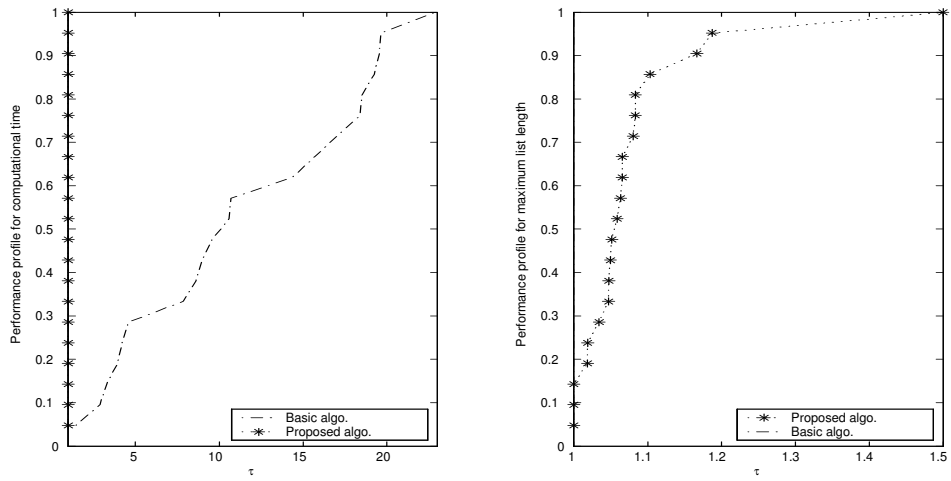


FIGURE 2.2. Performance profiles of time and maximum list length for  $\varepsilon = 10^{-6}$ .

TABLE 2.1. Performance comparison of algorithms for  $\epsilon = 10^{-4}$ .

S.No	Problem	$l$	Soln	Model Algorithm	Proposed Algorithm	Speed-up factor	Percentage reduction in time
1	Toolbox1	2	t	1.76	0.41	4.29	76.73%
			b	2	2		
			ml	13	12		
			fe	47	47		
			ge	47	47		
			flops	21,580	18,760		
2	Toolbox2	2	t	0.21	0.14	1.5	33.3%
			b	1	1		
			ml	3	2		
			fe	14	14		
			ge	7	7		
			flops	2886	2805		
3	Hansen1	2	t	1.63	0.52	3.13	68.1%
			b	2	2		
			ml	19	16		
			fe	117	117		
			ge	40	40		
			flops	17,950	16,590		
4	Lotka3	3	t	15.31	1.57	9.76	89.75%
			b	7	7		
			ml	88	84		
			fe	784	784		
			ge	295	295		
			flops	264,115	247,662		
5	Rediff3	3	t	1.37	0.48	3.4	70.8%
			b	2	2		
			ml	13	12		
			fe	76	76		
			ge	31	31		
			flops	25,007	23,867		
6	Redcyc4	4	t	286.29	13.88	20.63	95.15%
			b	5	5		
			ml	542	532		
			fe	9589	9589		



Table 2.1 (Contd.)

S.No	Problem	$l$	Soln	Model Algorithm	Proposed Algorithm	Speed-up factor	Percentage reduction in time
6	Redcyc4	4	ge	3458	3458		
			flops	5,724,494	5,466,165		
7	Lorentz4	4	t	15.65	3.76	4.16	75.97%
			b	2	2		
			ml	21	18		
			fe	723	723		
			ge	262	262		
			flops	370,842	354,689		
8	quad4	4	t	12.41	1.44	8.61	88.4%
			b	1	1		
			ml	67	62		
			fe	502	502		
			ge	183	183		
			flops	287,586	260,733		
9	Lotka5	5	t	677.83	34.5	19.65	94.91%
			b	3	3		
			ml	1072	1008		
			fe	14404	14404		
			ge	6239	6239		
			flops	17,955,022	16,451,452		
10	Eco5	5	t	19.34	2.2	8.79	88.62%
			b	2	2		
			ml	61	58		
			fe	634	634		
			ge	239	239		
			flops	568,753	568,753		
11	Wright5	5	t	17.71	1.99	8.89	88.76%
			b	6	6		
			ml	58	58		
			fe	628	628		
			ge	231	231		
			flops	527,881	507,976		

Table 2.1 (Contd.)

S.No	Problem	$l$	Soln	Model Algorithm	Proposed Algorithm	Speed-up factor	Percentage reduction in time
12	Redeco5	5		149.48	9.91	15.1	94%
			b	4	4		
			ml	128	116		
			fe	5744	5744		
			ge	2131	2131		
			flops	4,838,140	4,628,246		
13	Neuro6	6	t	325.21	14.33	22.71	95.6%
			b	1	1		
			ml	859	806		
			fe	7862	7862		
			ge	2899	2899		
			flops	11,645,809	10,930,525		
14	Trink	6	t	858.72	46.16	18.6	94.62%
			b	2	2		
			ml	1906	1816		
			fe	23782	23782		
			ge	7513	7513		
			flops	27,423,261	26,393,938		
15	Redeco6	6	t	91.32	5.57	16.39	93.9%
			b	4	4		
			ml	324	318		
			fe	2824	2824		
			ge	1109	1109		
			flops	3,685,274	3,568,688		
16	Hansens	6	t	14.08	3.33	4.22	76.38%
			b	1	1		
			ml	32	32		
			fe	225	225		
			ge	98	98		
			flops	404,998	394,139		

Table 2.1 (Contd.)

S.No	Problems	$l$	Soln	Model	Proposed	Speed-up	Percentage
				Algorithm	Algorithm	factor	reduction in time
17	Systems8	8	t	1277.4	84.49	15.11	93.4%
			b	4	4		
			ml	1151	1080		
			fe	30,743	30,743		
			ge	11,058	11,058		
			flops	72,860,411	71,374,087		
18	Robot	8	t	20.78	2.06	7.99	87.49%
			b	16	16		
			ml	56	56		
			fe	438	438		
			ge	155	155		
			flops	1,063,466	1,041,461		
19	Kinema	9	t	903.15	62.93	14.35	93.03%
			b	2	2		
			ml	964	932		
			fe	16434	16434		
			ge	6595	6595		
			flops	58,381,105	57,491,849		
20	Ku10	10	t	2615.7	332.27	7.87	87.3%
			b	2	2		
			ml	3295	3112		
			fe	44937	44937		
			ge	16678	16678		
			flops	179,009,004	176,506,127		
21	Sparsed1	12	t	305.71	16.21	18.86	94.7%
			b	1	1		
			ml	310	294		
			fe	4927	4927		
			ge	1838	1838		
			flops	32,894,146	32,526,802		
					Average	11.2	84.8%

TABLE 2.2. Performance comparison of algorithms for  $\varepsilon = 10^{-6}$ .

S.No	Problems	$l$	Soln	Model Algorithm	Proposed Algorithm	Speed-up factor	Percentage reduction in time
1	Toolbox1	2	t	1.79	0.39	4.59	78.21%
			b	2	2		
			ml	13	12		
			fe	47	47		
			ge	47	47		
			flops	21,580	18,760		
2	Toolbox2	2	t	0.22	0.15	1.5	31.82%
			b	1	1		
			ml	3	2		
			fe	14	14		
			ge	7	7		
			flops	2886	2805		
3	Hansen1	2	t	2.27	0.69	3.28	69.6%
			b	2	2		
			ml	19	16		
			fe	165	165		
			ge	56	56		
			flops	25,146	23,213		
4	Lotka3	3	t	16.36	1.53	10.69	90.65%
			b	7	7		
			ml	88	84		
			fe	792	792		
			ge	299	299		
			flops	267,422	250,754		
5	Rediff3	3	t	1.36	0.47	2.89	65.4%
			b	2	2		
			ml	13	12		
			fe	76	76		
			ge	31	31		
			flops	25,007	23,867		
6	Redcyc4	4	t	300.11	15.29	19.63	94.91%
			b	5	5		
			ml	542	532		
			fe	9941	9941		

Table 2.2 (Contd.)

S.No	Problems	$l$	Soln	Model Algorithm	Proposed Algorithm	Speed-up factor	Percentage reduction in time
6	Redcyc4	4	ge	3584	3584		
			flops	5,934,954	5,668,344		
7	Lorentz4	4	t	20.98	5.31	3.95	74.69%
			b	2	2		
			ml	21	18		
			fe	976	976		
			ge	355	355		
			flops	502,500	480,889		
8	quad4	4	t	12.91	1.5	8.61	88.4%
			b	1	1		
			ml	67	62		
			fe	976	976		
			ge	191	191		
			flops	299,306	271,526		
9	Lotka5	5	t	687.62	37.16	18.5	94.6%
			b	3	3		
			ml	1072	1008		
			fe	14902	14902		
			ge	6413	6413		
			flops	18,490,098	16,935,369		
10	Eco5	5	t	21.07	2.2	9.49	89.46%
			b	2	2		
			ml	61	58		
			fe	637	637		
			ge	240	240		
			flops	571,223	551,505		
11	Wright5	5	t	18.08	2.01	8.99	88.88%
			b	6	6		
			ml	58	58		
			fe	643	643		
			ge	238	238		
			flops	542,936	522,576		

Table 2.2 (Contd.)

S.No	Problems	$l$	Soln	Model Algorithm	Proposed Algorithm	Speed-up factor	Percentage reduction in time
12	Redeco5	5	t	209.97	13.37	15.7	93.63%
			b	4	4		
			ml	128	116		
			fe	7921	7921		
			ge	2938	2938		
			flops	6,673,526	6,383,285		
13	Neuro6	6	t	326.35	14.29	22.84	95.62%
			b	1	1		
			ml	859	806		
			fe	7866	7866		
			ge	2901	2901		
			flops	11,653,133	10,937,716		
14	Trink	6	t	865.38	47.03	18.4	94.57%
			b	2	2		
			ml	1906	1816		
			fe	23786	23786		
			ge	7515	7515		
			flops	27,430,080	26,400,529		
15	Redeco6	6	t	92.65	5.45	17	94.12%
			b	4	4		
			ml	324	318		
			fe	2832	2832		
			ge	1113	1113		
			flops	3,698,053	3,581,087		
16	Hansens	6	t	14.51	3.43	4.23	76.36%
			b	1	1		
			ml	32	32		
			fe	229	229		
			ge	100	100		
			flops	412,906	401,970		



# 3

## On speeding up the covering algorithm

### 3.1 Introduction

This chapter addresses the problem of finding in a given box all solutions of a nonlinear system with more variables than equations. This problem is clearly of a broad scope and has numerous applications in engineering and sciences.

The problem can sometimes be solved by one of the following methods [32] : (i) random search, (ii) an exhaustive grid search on the given box, (iii) more specialized or ad hoc methods, such as the Jenkins-Traub method for finding all roots of a single polynomial, and (iv) homotopy continuation methods [54]. The reader is referred to [32], [33] for a discussion and comparison of these methods.

In the frame work of interval analysis , Neumaier proposed the so-called covering algorithm [61] to solve the problem. Consider a finite-dimensional system of nonlinear equations of the form

$$f(x) = 0 \quad (3.1)$$

where  $f$  is a function defined on a subset  $\mathcal{D} \subseteq \mathbb{R}^l$  with values in  $\mathbb{R}^m$ ,  $m \leq l$ . If  $f$  is continuously differentiable in  $\mathcal{D}$  and  $f'(x)$  has rank  $m$  for all  $x$  in a neighborhood of the solution set

$$\mathcal{M} = \{x \in \mathcal{D} \mid f(x) = 0\}$$

then the solution set  $\mathcal{M}$  of (3.1) is a  $p$ -dimensional manifold in  $\mathbb{R}^l$ ,  $p = l - m$ . The vector  $x$  of variables often contains  $p$  distinguished variables, called as *parameters*.

We are interested in that part of  $\mathcal{M}$  for which all variables (and parameters) lie within certain bounds

$$\underline{\mathbf{X}}_i \leq x_i \leq \overline{\mathbf{X}}_i \quad (i = 1, \dots, l)$$



so that only the solutions of (3.1) contained in a box  $\mathbf{X}^0 \in I(\mathbb{R})^l$  are sought.

For any  $\mathbf{X} \in I(\mathcal{D})$ , define

$$\sum(F, \mathbf{X}) := \{x \in \mathbf{X} \mid f(x) = 0\}. \quad (3.2)$$

The covering algorithm of Neumaier [61] consists of covering the set  $\sum(F, \mathbf{X}^0)$  by a collection of smaller and smaller boxes which give increasingly accurate information about the location of the solution set. The algorithm uses the zero exclusion test and the generalized Gauss-Seidel method to discard irrelevant parts of  $\mathbf{X}^0$ . However, in each iteration of the covering algorithm, only *one* box is processed.

The aim of this chapter is to show that the covering algorithm can be speeded up by *several* orders of magnitude, if in each iteration of the algorithm, we simultaneously process *all* boxes that are present.

## 3.2 Neumaier's covering algorithm

We first outline Neumaier's covering algorithm to solve (3.1).

**Algorithm: Neumaier's covering algorithm** [61]

Inputs: the initial box  $\mathbf{X}^0$ , a continuous inclusion function (denoted as  $F$ ) of the given function  $f$ , and a parameter  $\varepsilon$  to check if the width of a box is small.

Begin Algorithm

1. Enter the initial box into the stack.
2. If the stack is empty, go to step 9.
3. Choose the first box from the stack.
4. Discard irrelevant parts of the box using the zero exclusion test and the GGS method (see Remarks 3.1 and 3.2 below).
5. If the box is empty, go to step 2.
6. If the width of the box is less than or equal to  $\varepsilon$ , print the box and go to step 2.
7. Bisect the box along the maximum width coordinate direction and enter the halved boxes into the stack at the end.
8. Go to step 2.
9. Stop.

End Algorithm.

**Remark 3.1** *If  $0 \notin F(\mathbf{X})$  then  $\mathbf{X}$  contains no solution point and is discarded (the zero exclusion test). Note that since  $F(\mathbf{X})$  generally overestimates the range, there may or may not be a solution point in  $\mathbf{X}$  if  $0 \in F(\mathbf{X})$ . In this case a more refined test is used as in the following remark.*

**Remark 3.2** *The algorithm is speeded up by finding a smaller box containing all solutions in  $\mathbf{X}$ , using the generalized Gauss-Seidel method (GGS) [61]. The GGS method is applied to the homogeneous linear interval equation*

$$\tilde{A}\tilde{D} = 0, \tilde{A} \in \mathbf{A}, \tilde{D} \in \mathbf{D} \quad (3.3)$$

where

$$\tilde{D} \in \mathbf{D} := \begin{pmatrix} \mathbf{X} - \tilde{z} \\ 1 \end{pmatrix}, \tilde{A} \in \mathbf{A} := (F'(\mathbf{X}), F(\tilde{z})); \quad (3.4)$$

and  $\tilde{z} \in \mathbf{X}$ . If this linear interval system is found incompatible then  $\mathbf{X}$  can be discarded. Else, the algorithm proceeds by replacing  $\mathbf{X}$  with  $\mathbf{X}' = \tilde{z} + \mathbf{D}'$ , where  $\mathbf{D}'$  is the solution constructed using the GGS method. To further speed up the algorithm, preconditioning of the linear system can be done, and interval slopes [41] instead of gradients used.

### 3.3 Proposed algorithm

Note that in each iteration of Neumaier's covering algorithm, only one box is processed. Such processing is inherently slow, due to its *sequential* nature. On the other hand, in each iteration of the covering algorithm, it is clearly possible to simultaneously process *all* boxes that are present (this can be done without altering in any way the essence of the algorithm). As will be seen below, the strategy results in greatly speeding up the algorithm, because all boxes present in every iteration are processed concurrently, i.e., in a *parallel* manner. We call this version of the covering algorithm as the *vectorized* covering algorithm.

**Algorithm: vectorized covering algorithm**

arises from Neumaier's covering algorithm by making the following changes.

- Step-3: Choose all the boxes present in the stack.
- Step-4: Discard irrelevant parts of all boxes:
  - (vectorized zero-exclusion test): using vectorized interval arithmetic operations, evaluate the interval extension  $F$  over all the boxes. Discard all those boxes for which  $0 \notin F(\mathbf{X})$ . If there are no more boxes remaining, go to step 9.
  - (vectorized GGS method) using vectorized gradient or slope evaluations, set up the  $\mathbf{A}, \mathbf{D}$  matrices in (3.3, 3.4) for all boxes. Using vectorized interval arithmetic operations, apply the GGS method simultaneously to all the resulting homogeneous linear interval equations, and obtain smaller boxes containing the solution points in the respective boxes.
- Step-5: Find and discard all empty boxes. If there are no more boxes remaining, go to step 9.

- Step-6: Using vectorized interval arithmetic operations, find the widths of all boxes. Then, find and print all those boxes satisfying the box-width condition (i.e., width of box  $\leq \varepsilon$ ). Discard the just printed boxes.
- Step-7: If there are no boxes remaining, go to step 9. Else, using vectorized interval arithmetic operations, find the maximum width coordinate directions for all boxes, and bisect simultaneously all the boxes along these (respective) directions. Enter all the resulting halved boxes into the stack.

**Remark 3.3** *It is emphasized that the formulation of the algorithm is independent of MATLAB / INTLAB, and it can be run on any computer that has an interval arithmetic compiler supporting vectorized interval arithmetic operations, such as Forte FORTRAN 95 [73].*

**Remark 3.4** *The vectorized covering algorithm does not require a parallel computer. That is, the vectorization is efficient even on serial architectures where vectorization brings advantages.*

**Remark 3.5** *It follows from a result in [57] that for  $\varepsilon > 0$ , the above algorithm terminates after at most  $\chi^n - 1$  iterations, where  $\chi = w(\mathbf{X}^0) / \varepsilon$ .*

### 3.4 Numerical results

We consider some examples for comparing the performance of the proposed vectorized covering algorithm with that of the covering algorithm. The examples are listed in Appendix.

All computations are carried out on a PC/Pentium-III 550 MHz machine with 384 MB RAM using INTLAB [68]. Tables 3.1 and 3.2 give the computational results for the various examples. The Tables list the number of boxes in the covering of the solution set, the execution time (seconds), and the number of floating operations (*flops*) taken. We used two values of  $\varepsilon$ , that is,  $\varepsilon = 0.01$  and  $0.001$ .

The following observations are made regarding the results given in the Tables :

1. The same number of covering boxes are obtained using either algorithm.
2. The proposed algorithm is faster than the covering algorithm in all examples. In all examples except example 3, the reduction in computational time is around 96 – 99%. In example 3, the same is around 30 – 40%.
3. The speed up is particularly attractive in those examples demanding large computational times using the covering algorithm. In such examples, the proposed algorithm is faster than the original one by up to 2 orders of magnitude.
4. The speed up factor gets better with accuracy.

5. The number of *flops* is less with the proposed algorithm in every example.

We conclude the chapter with some remarks:

**Remark 3.6** *The observed reduction in computational overhead (in terms of flops) can be attributed to the fact that in INTLAB, interval arithmetic and slopes are done by operator overloading, which incurs substantial overhead. This overhead is needed once for each box in the (sequential) covering algorithm, but only once per major iteration in the vectorized version [60].*

**Remark 3.7** *The actual speed up factor perhaps varies considerably with the computing environment used, and may be less conspicuous in programming languages where control structures are more efficiently implemented than in MATLAB.*

## 3.5 Conclusions

A vectorized version of Neumaier's covering algorithm was proposed for solving finite-dimensional systems of parameter - dependent nonlinear equations. It was demonstrated through several test examples that the vectorized algorithm is significantly faster (by up to 2 orders of magnitude in demanding problems) than the original algorithm. Moreover, it is noteworthy that the vectorized algorithm does not require a parallel computer or vector processors but can be run on any computer (such as a PC) that has an interval arithmetic compiler supporting vectorized interval arithmetic operations.

TABLE 3.1. Comparisons of algorithms for  $\varepsilon = 0.01$ .

S.No	Examples	m	$l$	Solutions	Covering algorithm	Proposed algorithm	Speed up factor
1	One dimensional manifold [61]	1	2	covering boxes	2406	2406	100.3
				time(s)	180.55	1.6	
				<i>flops</i>	1354502	1060178	
2	Tunneling diode [76]	1	2	covering boxes	1473	1473	97.7
				time(s)	122.13	1.25	
				<i>flops</i>	1051102	783391	
3	Combustion chemistry [53]	2	2	covering boxes	2	2	1.79
				time(s)	1.81	1.09	
				<i>flops</i>	13865	12947	
4	Hippopede [44]	2	3	covering boxes	896	896	68.89
				time(s)	84.05	1.22	
				<i>flops</i>	795304	558268	
5	PUMA robot [53]	8	8	covering boxes	28	28	25.83
				time(s)	105.39	4.08	
				<i>flops</i>	1351569	1193709	

TABLE 3.2. Comparisons of algorithms for  $\varepsilon = 0.001$ .

S.No	Examples	m	$l$	Solutions	Covering algorithm	Proposed algorithm	Speed up factor
1	One dimensional manifold [61]	1	2	covering boxes	26955	26955	313.01
				time(s)	3274.1	10.46	
				<i>flops</i>	14218727	11142507	
2	Tunneling diode [76]	1	2	covering boxes	17060	17060	272.75
				time(s)	1693.8	6.21	
				<i>flops</i>	10011584	7503713	
3	Combustion chemistry [53]	2	2	covering boxes	2	2	1.47
				time(s)	2.15	1.46	
				<i>flops</i>	17575	16567	
4	Hippopede [44]	2	3	covering boxes	6884	6884	189.57
				time(s)	635.07	3.35	
				<i>flops</i>	5231611	3685222	
5	PUMA robot [53]	8	8	covering boxes	40	40	27.06
				time(s)	125.3	4.63	
				<i>flops</i>	1635829	1444654	



# 4

## On speeding up the global optimization algorithm

### 4.1 Introduction

Let  $f : \mathbf{X} \subseteq \mathbb{R}^l \rightarrow \mathbb{R}$  be a differentiable function. Let  $\bar{f}(\mathbf{X})$  denote the set of all values of  $f$  on  $\mathbf{X}$ . We seek global optimization algorithms that are able to efficiently determine arbitrarily good lower bounds for the minimum of  $\bar{f}(\mathbf{X})$ .

Many algorithms based on interval analysis (IA) are available to solve this unconstrained global optimization problem, see for instance, [26], [34], [65]. A model or *basic*<sup>1</sup> branch and bound algorithm of IA consists of the Moore-Skelboe algorithm [51] augmented with the mid-point test of Ichida and Fuiji [30] and the monotonicity test detailed in [65]. Although this basic algorithm is *reliable*, it is sometimes found to be slow for ‘difficult’ problems. Therefore, several researchers recently proposed parallel processing techniques on multi-processor machines in order to speed up interval global optimization methods, especially for large-scale problems [7], [16], [50], [81]. However, as single processor machines such as desktop computers are more widely used, it would be advantageous to have techniques that speed up the algorithm on single processor machines.

In this chapter, we propose some modifications of the basic interval global optimization algorithm to speed up the algorithm for *single* processor machines. The proposed algorithmic changes are as follows.

- As is well known to interval analysts, in the basic algorithm we choose for processing *only* the first box from the list at each iteration. In the proposed modification, we choose and process *all* boxes from the list at each iteration.

---

<sup>1</sup>The basic or model version does not include local search procedures, concavity tests, and Newton-like steps, see [15], [65], [66].



- Using the fact that we are processing all the boxes simultaneously, we propose to use a new cut-off test (also called as the midpoint test) in the algorithm. In the basic algorithm, the cut-off value is the function value at the midpoint of the *leading* box at each iteration. In the proposed algorithm, the cut-off value is the minimum of the function values at the midpoints of *all* boxes in the list at each iteration. Hence, the new cut-off test is expected to discard more irrelevant boxes at each iteration.
- In the basic algorithm, the monotonicity test is applied to the subboxes obtained by bisection of the *leading* box at each iteration. In the proposed algorithm, the monotonicity test is applied to the subboxes obtained by bisection of *all* boxes in the list at each iteration.

Finally, we evaluate the performances of the proposed and basic algorithms on fifty standard optimization test functions given in [52] and [66].

## 4.2 Basic algorithm

### A Basic Algorithm of IA for Unconstrained Global Optimization [65], [66]

Inputs: The box  $\mathbf{X}$ , an inclusion function  $F$  (usually the natural interval extension, see [49]) for  $f : \mathbf{X} \rightarrow \mathbb{R}$ , and accuracy parameter  $\varepsilon_F$ .

BEGIN Algorithm

1. Set  $\mathbf{Z}_1 = \mathbf{X}$ , calculate  $F(\mathbf{Z}_1)$ , and set  $z_1 = \min F(\mathbf{Z}_1)$ . Next, initialize list  $L = ((\mathbf{Z}_1, z_1))$  and the cut-off value  $c = f(m(\mathbf{Z}_1))$ .
2. Choose a coordinate direction  $k$  parallel to which  $\mathbf{Z}_1$  has an edge of maximum length.
3. Bisect  $\mathbf{Z}_1$  in direction  $k$  getting boxes  $\mathbf{V}^1$  and  $\mathbf{V}^2$  such that  $\mathbf{Z}_1 = \mathbf{V}^1 \cup \mathbf{V}^2$ .
4. Calculate  $F(\mathbf{V}^1)$  and  $F(\mathbf{V}^2)$ , and set  $v^1 = \min F(\mathbf{V}^1)$ ,  $v^2 = \min F(\mathbf{V}^2)$ .
5. Remove  $(\mathbf{Z}_1, z_1)$  from the list  $L$ .
6. Discard the pair  $(\mathbf{V}^i, v^i)$  if  $v^i > c$ , where  $i \in \{1, 2\}$ .
7. (Monotonicity test, see Remark 4.2) discard the remaining pair  $(\mathbf{V}^i, v^i)$  if  $0 \notin F'_j(\mathbf{V}^i)$  for any  $j \in \{1, 2, \dots, l\}$ , and  $i = 1, 2$ .
8. Add the remaining pair(s) to  $L$ . If  $L$  is empty, then EXIT. Otherwise, arrange  $L$  such that the second members of all pairs of  $L$  do not decrease, and denote the pairs as in Remark 4.1. Choose the first item  $(\mathbf{Z}_1, z_1)$ .
9. Update the cut-off value as  $c = \min \{c, f(m(\mathbf{Z}_1))\}$ .

10. (Cut-off test) discard from  $L$  all pairs whose second members are greater than the cut-off value  $c$ .
11. If the termination criterion holds (see Remarks 4.3 and 4.4) then EXIT algorithm.
12. Go to Step 2.

END Algorithm

**Remark 4.1** Let  $l_r$  denote the current list length. Then, the items in list  $L$  at iteration  $r$  consists of pairs denoted as  $(\mathbf{Z}_1, z_1), \dots, (\mathbf{Z}_{l_r}, z_{l_r})$ , where,  $z_n = \min F(\mathbf{Z}_n)$ ,  $n = 1, \dots, l_r$ .

**Remark 4.2** In the monotonicity test, if  $0 \notin F'_j(\mathbf{V}^i)$  then the interior of  $\mathbf{V}^i$  cannot contain a global minimizer. The edge of  $\mathbf{V}^i$  still can contain global minimizer if that part of the edge which has the smallest function values is also part of the edge of  $\mathbf{X}$ . Otherwise, no global minimizer lies in  $\mathbf{V}^i$ . For details, see [65].

**Remark 4.3** We use two different termination criteria for evaluating the performances of the algorithms:

- Termination criterion A: If  $w(F(\mathbf{Z}_1)) < \varepsilon_F$ . This termination criterion gives the global minimum and a minimizer in the given domain.
- Termination criterion B: If

$$\begin{aligned} \max \{w(F(\mathbf{Z}_1)), \dots, w(F(\mathbf{Z}_{l_r}))\} &< \varepsilon_F, \text{ and} \\ \max \{w(\mathbf{Z}_1), \dots, w(\mathbf{Z}_{l_r})\} &< \varepsilon_X. \end{aligned}$$

This termination criterion gives the global minimum and all the minimizers in the given domain.

**Remark 4.4** In case of termination criterion A, we print 'computed global minimum is  $= z_1$ , before exiting. In case of termination criterion B, we print the items in list  $L$  before exiting.

### 4.3 Proposed algorithm

In the basic algorithm described above, in each iteration we choose only the leading box from the list  $L$  for processing. On the other hand, in the proposed algorithm given below we choose *all* boxes from the list for processing. To perform function and gradient evaluations, monotonicity test, midpoint test, width checks, and bisections on *all* boxes in an iteration, we use *vectorized* interval arithmetic operations. We next present the proposed algorithm.

#### Proposed Algorithm for Global Optimization

BEGIN Algorithm

1. Set  $\mathbf{Z}_1 = \mathbf{X}$ , calculate  $F(\mathbf{Z}_1)$ , and set  $z_1 = \min F(\mathbf{Z}_1)$ . Next, initialize list  $L = ((\mathbf{Z}_1, z_1))$  and the cut-off value  $c = f(m(\mathbf{Z}_1))$ .
2. Set  $l_r$  as the length of  $L$ . Then, choose a coordinate direction  $k_n$  parallel to which  $\mathbf{Z}_n$  has an edge of maximum length,  $n = 1, \dots, l_r$ .
3. Bisect  $\mathbf{Z}_n$  in direction  $k_n$  getting boxes  $\mathbf{V}_n^1$  and  $\mathbf{V}_n^2$  such that  $\mathbf{Z}_n = \mathbf{V}_n^1 \cup \mathbf{V}_n^2$ ,  $n = 1, \dots, l_r$ .
4. Calculate  $F(\mathbf{V}_n^1)$  and  $F(\mathbf{V}_n^2)$ , and set  $v_n^i = \min F(\mathbf{V}_n^i)$ , for  $i = 1, 2$ , and  $n = 1, \dots, l_r$ .
5. Discard the pair  $(\mathbf{V}_n^i, v_n^i)$  if  $v_n^i > c$ , where  $i \in \{1, 2\}$ ,  $n \in \{1, \dots, l_r\}$ .
6. (Monotonicity test) Discard the remaining pairs  $(\mathbf{V}_n^i, v_n^i)$  if  $0 \notin F_j'(\mathbf{V}_n^i)$  for any  $j \in \{1, 2, \dots, l\}$ , and  $i = 1, 2, n = 1, \dots, l_r$ .
7. Delete all items from  $L$ , and enter the remaining pair(s) of above step in  $L$ . Set  $l_{r'}$  as the (temporary) length of  $L$ . If  $l_{r'}$  is zero, then EXIT. Otherwise, arrange  $L$  such that the second members of all pairs of  $L$  do not decrease, and denote the pairs as in Remark 4.1.
8. Update the cut-off value as  $c = \min \{c, f(m(\mathbf{Z}_1)), \dots, f(m(\mathbf{Z}_{l_{r'}}))\}$ .
9. (Cut-off test) Discard from  $L$  all pairs whose second members are greater than the cut-off value  $c$ .
10. If the termination criterion holds (c.f Remark 4.3 and 4.4) then EXIT algorithm.
11. Go to Step 2.

END Algorithm

#### 4.4 Test results

For evaluating the effectiveness of the proposed algorithm, we consider the *full* benchmark suite of 35 optimization test functions given in [52]. We also select additional 15 test functions from the collection of 39 test functions in [66]. The criteria we use for this selection are:

1. Avoid test functions that have already been considered in the suite in [52].
2. Choose one problem from each group of ‘similar’ test functions. For example, 12 functions called *Levy* have been put into 4 groups of ‘similar’ functions by Ratz and Csendes in [66]. By ‘similar’, we mean that the functions are identical except for dimensionality. Hence, we randomly choose one function from each of these groups. We similarly choose

among the test functions called as *Griewank*, *Ratz*, *Schwefel*, etc., to arrive at a total of fifteen test functions from this collection.

In test functions 1-35 from [52], given  $f_i : \mathbb{R}^l \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , with  $m \geq l$ , our aim is to find

$$\min \left\{ \sum_{i=1}^m f_i^2(x) : x \in \mathbf{X} \right\}$$

In the test functions 36-50 from [66], our aim is to find

$$\min_{x \in \mathbf{X}} f(x)$$

where the objective function  $f : \mathbb{R}^l \rightarrow \mathbb{R}$  is continuously differentiable and  $\mathbf{X} \subseteq \mathbb{R}^l$ .

For most test functions from [52], we select the initial domain  $\mathbf{X}$  as per Hansen [26, pp.135-136]. For the rest, with this initial domain selection both the algorithms fail, due to excessive time (greater than 10 hours) and / or memory requirements. In such test functions, we choose smaller initial domains, such that they include all the starting points given in [52]. For the test functions from [66], the domain is kept the same as given in [66]. We choose the accuracies as  $\varepsilon_F = 10^{-2}$  or  $10^{-4}$ .

We carry out all computations on a single processor PC/Pentium III 800 MHz machine with 512 MB RAM using version 3 of INTLAB [68]. Table 4 gives the domain used, the global minimum in the given domain and the minimizer for each test function.

To compare the performances of the algorithms, we use the following performance metrics

- Number of functional evaluations
- Number of floating operations (*flops*)
- Computational time, seconds
- Maximum list length

Tables 4.7 to 4.10 give the obtained results in terms of these performance metrics for the various test functions<sup>2</sup>. For each metric, in the last two columns of Tables 4.7-4.10 we give the values of ratio and the percent reduction computed as

$$\begin{aligned} \text{Ratio} &= \frac{\text{Perf. metric with basic algorithm}}{\text{Perf. metric with proposed algorithm}} \\ \text{Percent reduction} &= \frac{\text{Perf. metric with basic algorithm} - \text{Perf. metric with proposed algorithm}}{\text{Perf. metric with basic algorithm}} \times 100 \end{aligned}$$

Based on the data in the Tables, we compare the performance of the two algorithms using different evaluation methods: ranking, statistical measures, average and other measures, and

---

<sup>2</sup>A '\*' entry in the last two columns of Tables 4.7 – 4.14 indicates that a solution could not be obtained by the basic algorithm for the prescribed accuracy, due to excessive time requirements (greater than 10 hours).

performance profiles. These various methods are incorporated in the analysis to avoid dominance of any one test function on the final conclusions about the relative performance of the algorithms.

#### 4.4.1 Termination criterion A

##### Ranking

*Ranking* of the algorithms has been used for performance comparison, see for instance, [11], [55], [77]. For a given performance metric, ranking is based on the number of times a rule comes in the  $r^{th}$  place, here  $r = 1, \dots, n_s$ . A higher rank is assigned to the rule with lesser performance value, while we assign the same rank to more than one rule if they achieve the same performance value. Further, we assign the last rank to a rule if it is found unable to solve the problem due to excessive memory requirements. Table 4.1 gives the ranking of the algorithms in our studies.

At the outset, note that for the considered domains and accuracy, the proposed algorithm is able to solve all the test functions, whereas the basic algorithm is able to solve only 92% of the test functions. Table 4.1 shows that in a majority of the solved test functions, the proposed algorithm is better in terms of computational time and maximum list length metrics (six test functions require same number of function calls, four test functions require same *flops* and maximum list length with both the algorithms, and so both the algorithms obtain the first rank in these functions). Especially, the proposed algorithm is able to achieve the 1<sup>st</sup> rank in 94% of the test functions for the computational time metric.

##### Statistical measures

Next, we compare the performance of the algorithms based on the *distribution* of the difference between the performance metrics. Such a comparison has been done, for instance, in [11]. The minimum, first quartile, median, third quartile and maximum of this distribution are reported in Table 4.2. A positive value of the median for computational time and maximum list length indicates that proposed algorithm requires less computational time and maximum list length for more than half of the solved test functions. A negative value of median for the function evaluation and *flops* show that proposed algorithm requires more function evaluations for more than half of the test functions. The inter-quartile distance clearly shows the advantage of the proposed algorithm in terms of computational time and maximum list length metrics.

##### Minimum, mean, and maximum measures

In Table 4.3, we give the average (over all test functions) of the ratio and percent reduction defined earlier. Table 4.3 shows that with the proposed algorithm, on the average we obtain an increase in the number of function evaluations *flops* and maximum list lengths. Moreover, we

obtain a good reduction in the computational time with the proposed algorithm in majority of the test functions.

#### Performance profiles

The following observations are made from the performance profile plots computed for various performance metrics.

##### *Number of function evaluations*

Performance profile plots for the number of function evaluations (see Figure 4.1 ) show that the proposed algorithm solves only 20% of the (number of) test functions for  $\tau = 1$  and remaining 80% for  $\tau < 90$ . The basic algorithm is able to solve 92% of the test functions for  $\tau = 1$ . This shows the proposed algorithm is able to solve all the test functions for  $\tau < 90$ , and that it requires more number of function evaluations for 80% of the test functions.

##### *Computational effort (flops)*

Performance profile plots for computational effort in terms of *flops* are shown in Figure 4.2. The proposed algorithm solves 32% of the test functions for  $\tau = 1$ , and the remaining 78% for  $\tau < 80$ . The basic algorithm could solve 76% of the test functions for  $\tau = 1$ , and 16% of the test functions for  $\tau < 2$ . This shows the proposed algorithm is able to solve all the test functions for  $\tau < 80$ , and that 78% of the test functions require more computational effort with it.

*Computational time.* Performance profile plots for computational time (see Figure 4.3 ) show that the proposed algorithm is able to solve 94% of the test functions for  $\tau = 1$ . The basic algorithm is able to solve 6% of the test functions for  $\tau = 1$  and the remaining 86% for  $\tau < 500$ . In a majority of the test functions, the proposed algorithm is considerably faster.

##### *Maximum list length*

Performance profile plots for maximum list length (see Figure 4.4) show that the proposed algorithm solves 62% of test functions for  $\tau = 1$ , and remaining 38% for  $\tau < 65$ . The basic algorithm solves 46% for  $\tau = 1$ , and remaining 46% of test functions for  $\tau < 5$ . This shows that the proposed algorithm is able to solve all the test function for  $\tau < 65$ , and that it requires less memory for 62% of the test functions.

#### 4.4.2 Termination criterion B

Tables 4.11 - 4.14 give the obtained results in terms of the performance metrics for the various test functions when the algorithm is tested with termination B. At the outset, note that for the considered domains and accuracy, the proposed algorithm is able to solve all the test functions whereas the basic algorithm is able to solve only 80% of the test functions.

### Ranking

Table 4.4 gives the ranking of the algorithms in our studies (A higher rank is assigned to the algorithm with lesser performance metric value). Table 4.4 shows that in a majority of the solved test functions, the proposed algorithm is better in terms *flops*, computational time and maximum list length (two of the test function require same number of function calls and five of the test function require same maximum list length with either algorithm, and they get both ranks). Especially, the proposed algorithm is able to achieve the 1<sup>st</sup> rank in 94% of the test functions for the computational time metric.

### Statistical measures

Next, we compare the performance of the algorithms based on the *distribution* of the difference between the performance metrics. The minimum, first quartile, median, third quartile and maximum of this distribution are reported in Table 4.5.

A positive value of the median for *flops*, computational time and maximum list length indicates that proposed algorithm requires less *flops*, computational time and maximum list length for more than half of the solved test functions. A negative value of median for the function evaluation shows that proposed algorithm requires more function evaluations for more than half of the test functions. The inter-quartile distance clearly shows the advantage of the proposed algorithm in terms of *flops*, computational time and maximum list length.

### Minimum, mean, and maximum measures

In Table 4.6, we give the average (over all test functions) of the ratio and percent reduction defined earlier. Table 4.6 shows that with the proposed algorithm, on the average ratio we obtain decrease in the number of function evaluations, *flops*, computational time and maximum list length but average percent reduction shows only decrease in computational time.

### Performance profiles

The following observations are made from the performance profile plots computed for various performance metrics.

#### *Number of function evaluations*

Performance profile plots for the number of function evaluations (cf. Figure 4.5) show a better performance with the proposed algorithm. The proposed algorithm solves 52% of the (number of) test functions for  $\tau = 1$  and remaining 48% for  $\tau < 4.7$ . The basic algorithm is able to solve 52% of the test functions for  $\tau = 1$  and 38% of the test functions for  $\tau < 10$ . This shows that the proposed algorithm is able to solve all the test functions for  $\tau < 4.7$ , and that it requires less number of function evaluations for 52% of the test functions.

*Computational effort (flops).* Performance profile plots for computational effort in terms of *flops* are shown in Figure 4.6. The proposed algorithm solves 30% of the test functions for  $\tau = 1$ , and the remaining 70% for  $\tau < 4.75$ . The basic algorithm could solve 20% of the test functions for  $\tau = 1$ , and 60% of the test functions for  $\tau < 1250$ . This shows the proposed algorithm is able to solve all the test functions for  $\tau < 4.75$ , and that 30% of the test functions require less computational effort with it.

#### *Computational time*

Performance profile plots for computational time (cf. Figure 4.7) show the proposed algorithm is able to solve 94% of the test functions for  $\tau = 1$  and the remaining 6% for  $\tau < 1.23$ . The basic algorithm is able to solve only 80% of the test functions. The basic algorithm solves 6% of the test function for  $\tau = 1$  and the remaining 74% for  $\tau < 8729$ . In 94% test functions, the proposed algorithm is considerably faster.

#### *Maximum list length*

Performance profile plots for maximum list length (cf. Figure 4.8) show that the proposed algorithm solves 72% of test functions for  $\tau = 1$ , and remaining 28% for  $\tau < 4.7$ . The basic algorithm solves just 19% for  $\tau = 1$ , 6 and the remaining 61% for  $\tau < 8.1\%$ . This shows that the proposed algorithm is able to solve all the test function for  $\tau < 4.7$ , and that it requires less memory for 72% of the test functions.

## 4.5 Conclusions

### *Termination criterion A*

At the outset, we note that for the considered domains and accuracy, the proposed algorithm is able to solve *all* the test functions, whereas the basic algorithm is able to solve only about 92% of the test functions.

On an average, the proposed algorithm requires 489% more function evaluations, 538% more computational effort, 241% more list length. The reasons for this increase are as follows.

- The number of function evaluations and *flops* is more with the proposed algorithm because the function is evaluated on *all* boxes present in the list. Whereas in the basic algorithm, the function is evaluated only on the leading box of the list.
- The maximum list length is more with the proposed algorithm because if  $l_r$  boxes are present in the list at some iteration, then the number of boxes after bisection in the proposed algorithm is  $2l_r$  whereas it is only  $l_r + 1$  in the basic algorithm.

However, despite the increase in number of function evaluations, computational effort, and maximum list length, the proposed algorithm gives an average speed improvement of 68%.



Further, while improvements in speed are obtained in 94% test functions, improvements (i.e., reductions) in number of function evaluations, flops, and maximum list length are obtained in 20%, 32%, and 62% of test functions, respectively. The performance profile plots show a clear superiority of the proposed algorithm over the basic algorithm for the computational time.

In short, the proposed algorithm uses the extra function evaluations, computational effort, and maximum list length *effectively*, yielding considerable savings in computational time

#### ***Termination criterion B***

For the considered domains and accuracy, the proposed algorithm is able to solve all the test functions whereas the basic algorithm is able to solve only about 80% of the test functions.

On the average the proposed algorithm requires 66.37% more function evaluations, 49% more computational effort, 18% more memory but gives a speed improvement of 74.2%.

While the improvements in speed are obtained in 94% of test functions, improvements (i.e., reductions) in number of function evaluations, flops, and maximum list length are obtained in 52%, 60%, and 72% of test functions respectively.

The performance profile plots show a clear superiority of the proposed algorithm over the basic algorithm w.r.t. all the performance metrics considered for this termination criterion.

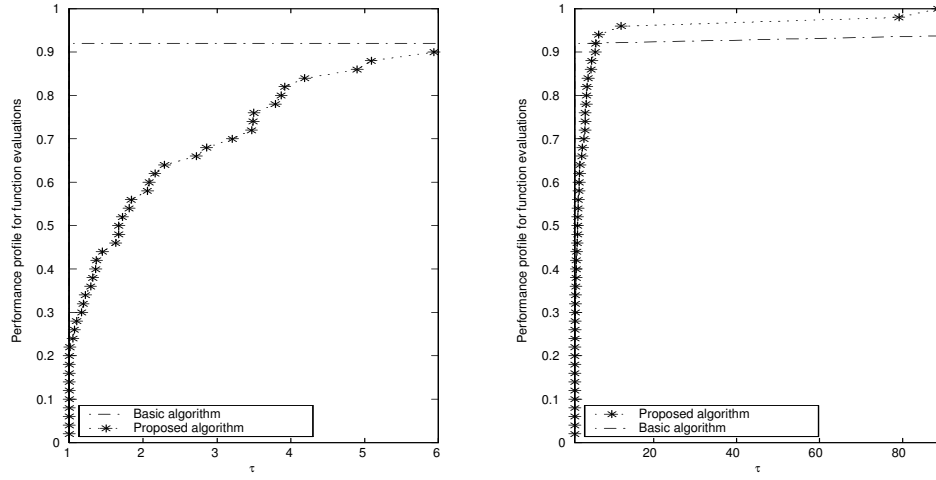


FIGURE 4.1. Performance profile plots for number of function evaluations with termination criterion A.

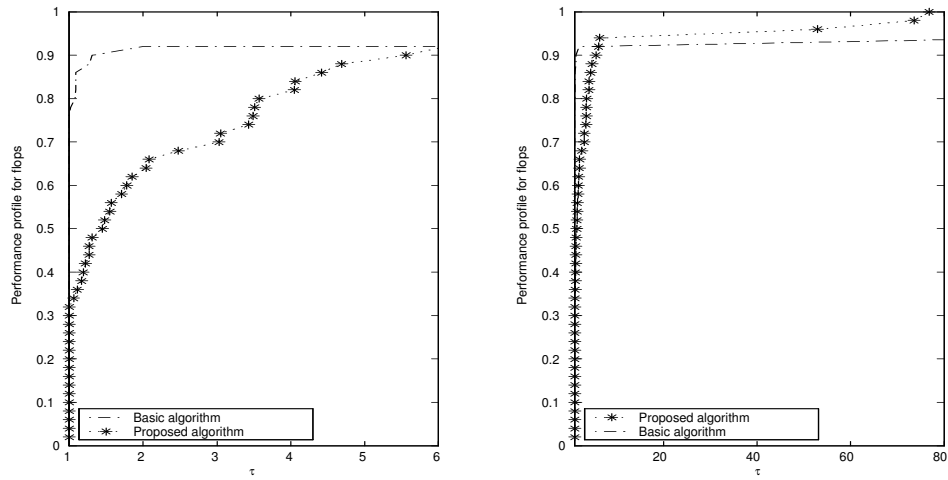


FIGURE 4.2. Performance profile plots for *flops* with termination criterion A.

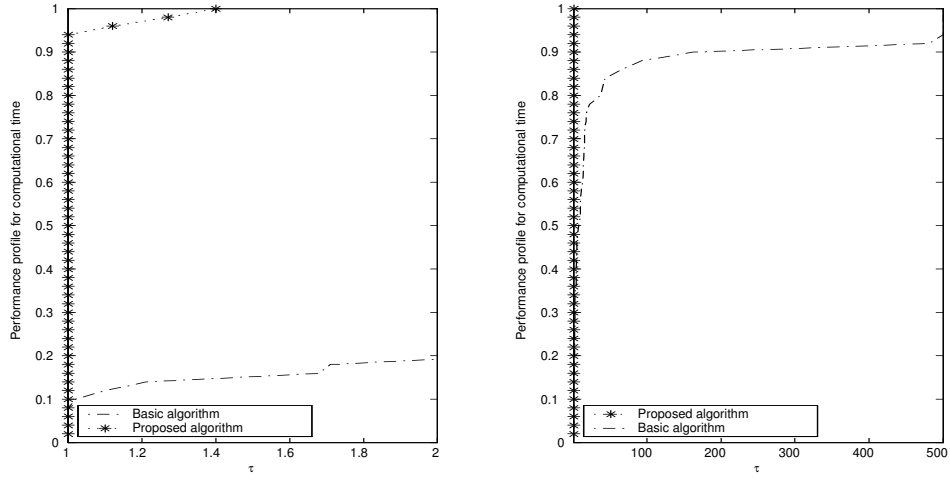


FIGURE 4.3. Performance profile plots for computational time (seconds) with termination criterion A.

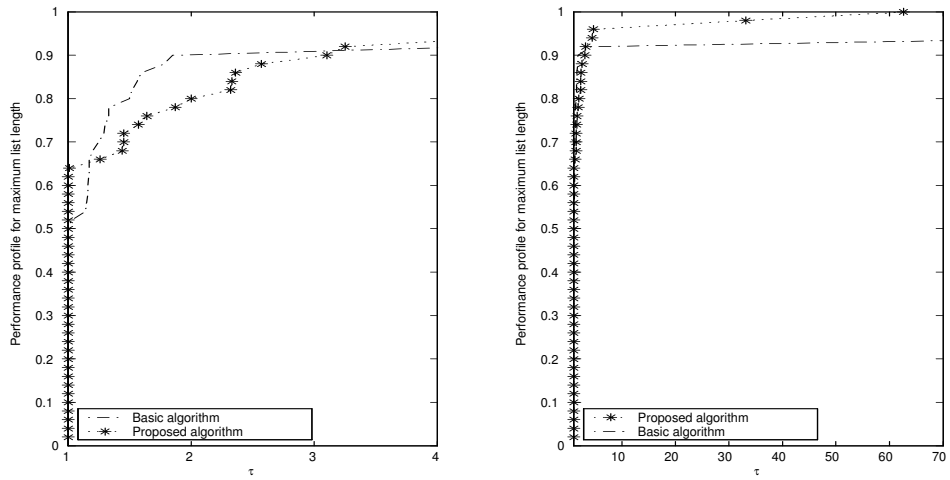


FIGURE 4.4. Performance profile plots for maximum list length with termination criterion A.

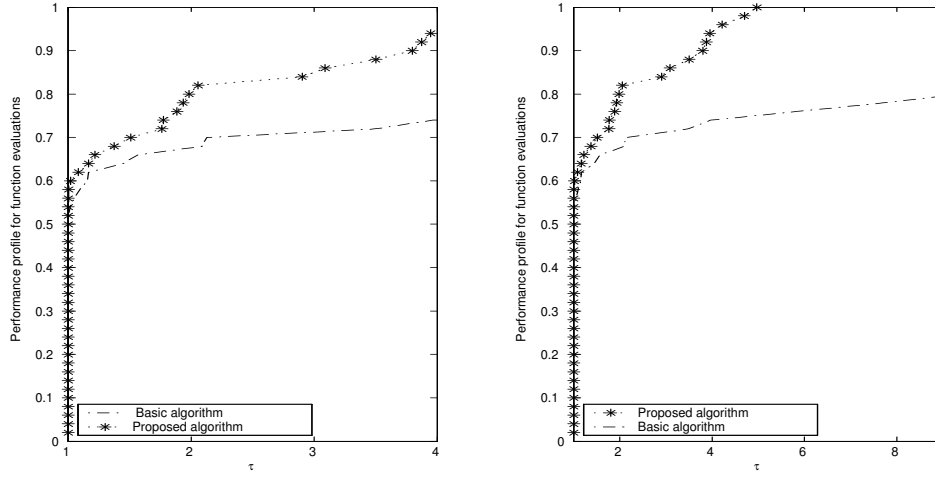
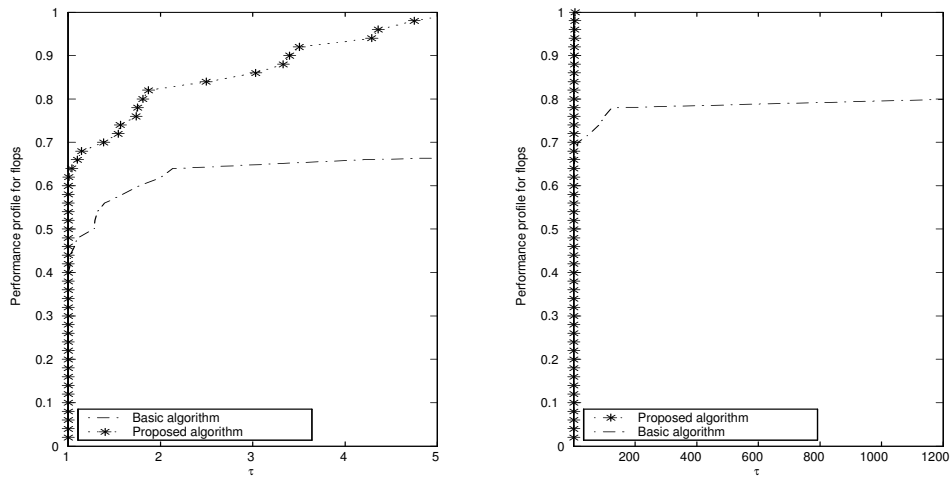


FIGURE 4.5. Performance profile plots for number function evaluations with termination criterion B.

FIGURE 4.6. Performance profile plots for *flops* with termination criterion B.

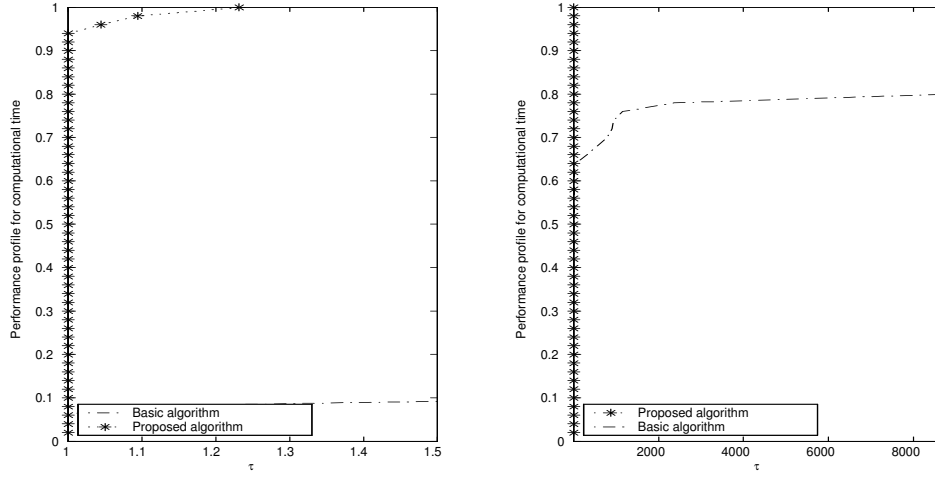


FIGURE 4.7. Performance profile plots for computational time with termination criterion B.

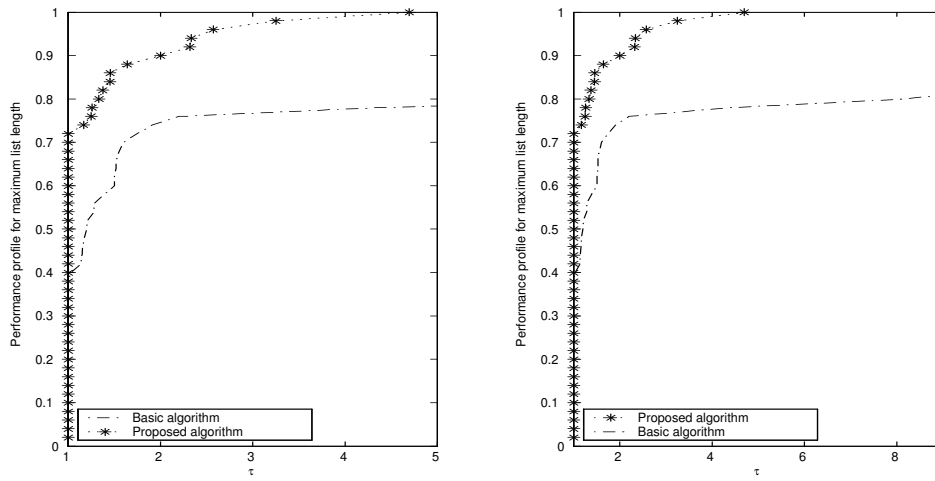


FIGURE 4.8. Performance profile plots for maximum list length with termination criterion B.

TABLE 4.1. Rankings with termination criterion A. Note that six test functions require same number of function calls, while four test functions require same flops and maximum list length with both the algorithms. Hence, both algorithms obtain the first rank in these functions.

Performance metric	Number of first ranks with basic algorithm	Number of first ranks with proposed algorithm
Function evaluations	46	10
Flops	38	16
Computational time	3	47
Maximum list length	23	31

TABLE 4.2. Statistical measures with termination criterion A.

Performance metric	Minimum	First Quartile	Median	Second Quartile	Maximum
Function evaluations	$-4.04 \times 10^5$	$-3.74 \times 10^3$	-439	-42	$6.88 \times 10^4$
Flops	$-1.16 \times 10^9$	$-2.59 \times 10^6$	$-1.14 \times 10^5$	$1.24 \times 10^4$	$2.94 \times 10^9$
Computational time	-29.7	0.87	8.95	$3.07 \times 10^2$	$2.1 \times 10^4$
Maximum list length	$-2.74 \times 10^4$	-23.75	0	13	2678

TABLE 4.3. Minimum, Mean, and Maximum of Ratios and Percent Reductions with termination criterion A.

Performance metric	Ratio			Percent Reduction		
	Min.	Mean	Max.	Min.	Mean	Max.
Function Evaluations	0.01	0.61	2.29	-8748%	-487.54%	56.38%
Flops	0.01	0.72	2.8	-7585.39%	-537.16%	64.3%
Computational time	0.71	26.08	481.71	-40%	68.44%	99.79%
Max. list length	0.02	0.98	4.31	-6163.16%	-240.99%	76.78%

TABLE 4.4. Rankings with termination criterion B.

Performance metric	Number of first ranks with basic algorithm	Number of first ranks with proposed algorithm
Function evaluations	26	26
Flops	20	30
Computational time	3	47
Maximum list length	19	36

TABLE 4.5. Statistical measures with termination criterion B.

Performance metric	Minimum	First Quartile	Median	Second Quartile	Maximum
Function evaluations	-2432	-393	-44	1071	534138
Flops	$-2.74 \times 10^7$	$-1.7 \times 10^5$	3326	$1.5968 \times 10^7$	$1.118 \times 10^{10}$
Computational time	-0.8	2.45	15.07	1133.2	$3.59 \times 10^4$
Maximum list length	-10923	-4.75	1.5	44	38125

TABLE 4.6. Minimum, Mean, and Maximum of Ratios and Percent Reductions with termination criterion B.

Performance Metric	Ratio			Percent Reduction		
	Min.	Mean	Max.	Min.	Mean	Max.
Function Evaluations	0.2	1.46	9.34	-397%	-52.38%	89.29%
Flops	0.19	41.35	1235.32	-429%	-49%	99.92%
Computational time	0.81	400.85	8744	-23.15%	74.17%	99.99%
Max. list length	0.21	1.32	8.08	-369.44%	-18.23%	87.63%

TABLE 4.7. Function evaluations for basic and proposed algorithms with termination criterion A.

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1	Rosenbrock	$l = 2, m = 2$	$10^{-4}$	116	590	0.2	-408.62%
2	Freudenstein and Roth	$l = 2, m = 2$	$10^{-4}$	1,376	2,244	0.61	-61.63%
3	Powell badly scaled	$l = 2, m = 2$	$10^{-4}$	$> 267,062$	7,704,142	*	*
4	Brown badly scaled	$l = 2, m = 3$	$10^{-4}$	377	33,357	0.01	-8748%
5	Beale	$l = 2, m = 3$	$10^{-4}$	440	600	0.73	-36.36%
6	Jenrich and Sampson	$l = 2, m = 10$	$10^{-4}$	3,340	3,344	0.99	-0.12%
7	Helical valley	$l = 3, m = 3$	$10^{-4}$	340	366	0.93	-7.65%
8	Bard	$l = 3, m = 15$	$10^{-4}$	54,326	70,068	0.78	-28.98%
9	Gaussian	$l = 3, m = 16$	$10^{-4}$	256	1,556	0.16	-507.81%
10	Meyer	$l = 3, m = 15$	$10^{-2}$	68,380	73,495	0.93	-7.48%
11	Gulf research	$l = 3, m = 3$	$10^{-4}$	$> 444,022$	5,686,389	*	*
12	Box three dimensional	$l = 3, m = 3$	$10^{-4}$	72,508	131,311	0.55	-81.10%
13	Powell singular	$l = 4, m = 4$	$10^{-4}$	5,228	5,222	1.00	11.48%
14	Wood	$l = 4, m = 6$	$10^{-4}$	320	1,900	0.17	-493.75%
15	Kowalik and Osborne	$l = 4, m = 11$	$10^{-4}$	160,352	333,996	0.48	-108.29%
16	Brown and Dennis	$l = 4, m = 20$	$10^{-4}$	36,386	36,750	0.99	-1.00%
17	Osborne 1	$l = 5, m = 33$	$10^{-4}$	39,362	47,090	0.84	-19.63%
18	Biggs EXP 6	$l = 6, m = 13$	$10^{-2}$	163,036	566,910	0.29	-247.72%
19	Osborne 2	$l = 11, m = 65$	$10^{-2}$	122,041	53,232	2.29	56.38%
20	Watson	$l = 6, m = 31$	$10^{-2}$	37,024	61,915	0.59	-67.23%
21	Extended Rosenbrock	$l = 4, m = 4$	$10^{-4}$	194	1,308	0.15	-574.23%
22	Extended Powell singular	$l = 4, m = 4$	$10^{-4}$	2,610	2,610	1.00	0.00%



Table 4.7 (Contd.)

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
23	Penalty I	$l = 4, m = 5$	$10^{-4}$	28,996	142,210	0.20	-390.45%
24	Penalty II	$l = 4, m = 8$	$10^{-4}$	3,406	269,458	0.01	-7811.30%
25	Variably dim.	$l = 2, m = 4$	$10^{-4}$	38	70	0.54	-84.21%
26	Trigonometric	$l = 4, m = 4$	$10^{-4}$	882	882	1.00	0.00%
27	Brown almost linear	$l = 4, m = 4$	$10^{-4}$	87,348	115,586	0.76	-32.33%
28	Discrete boundary value	$l = 2, m = 2$	$10^{-4}$	80	116	0.69	-45%
29	Discrete integral eq.	$l = 4, m = 4$	$10^{-4}$	80	228	0.35	-185%
30	Broyden tridiagonal	$l = 4, m = 4$	$10^{-4}$	594	2,492	0.24	-319.53%
31	Broyden branded	$l = 4, m = 4$	$10^{-4}$	396	814	0.49	-105.56%
32	Linear full rank	$l = 4, m = 4$	$10^{-4}$	1,070	3,430	0.31	-220.56%
33	Linear rank-1	$l = 3, m = 4$	$10^{-2}$	>189,548	1,062,802	*	*
34	Linear rank-1 with zero column and rows	$l = 2, m = 4$	$10^{-4}$	>249,718	1,572,820	*	*
35	Chebyquad	$l = 2, m = 2$	$10^{-4}$	198	242	0.82	-22.22%
36	SHCB: Six hump camel	$l = 2, m = 1$	$10^{-4}$	1,446	1,450	0.99	-0.28%
37	THCB: Three hump camel	$l = 2, m = 1$	$10^{-4}$	774	810	0.96	-4.65%
38	BR: Branin	$l = 2, m = 1$	$10^{-4}$	110	184	0.59	-67.27%
39	L3: Levy	$l = 2, m = 1$	$10^{-4}$	1,754	3,020	0.58	-72.17%
40	L5: Levy	$l = 2, m = 1$	$10^{-4}$	394	1546	0.26	-292.39%
41	L8: Levy	$l = 3, m = 1$	$10^{-4}$	62	240	0.26	-287.09%
42	L18: Levy	$l = 7, m = 1$	$10^{-4}$	184	644	0.29	-250%
43	Schw2.5: Booth	$l = 2, m = 1$	$10^{-4}$	146	200	0.73	-36.99%
44	Schw2.18: Matyas	$l = 2, m = 1$	$10^{-4}$	1,138	1,326	0.86	-16.52%
45	Schw3.1: Schwefel	$l = 3, m = 1$	$10^{-4}$	78	296	0.26	-279.49%
46	Schw3.1p: Schwefel	$l = 3, m = 1$	$10^{-4}$	230	804	0.29	-249.57%
47	Schw3.2: Schwefel	$l = 3, m = 1$	$10^{-4}$	186	402	0.46	-116.13%
48	Griew5: Griewank	$l = 5, m = 1$	$10^{-4}$	1904	5182	0.37	-172.16%
49	GP: Goldstein and price	$l = 2, m = 1$	$10^{-4}$	88,372	88,388	0.99	-0.02%
50	R4: Ratz	$l = 2, m = 1$	$10^{-4}$	1,294	1,574	0.82	-21.64%

TABLE 4.8. *Flops* for basic and proposed algorithms with termination criterion A.

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1	Rosenbrock	$l = 2, m = 2$	$10^{-4}$	11,916	52,717	0.23	-342.4%
2	Freudenstein and Roth	$l = 2, m = 2$	$10^{-4}$	233,610	346,128	0.67	-48.16%
3	Powell badly scaled	$l = 2, m = 2$	$10^{-4}$	$> 88,575,148$	$2.41 \times 10^9$	*	*
4	Brown badly scaled	$l = 2, m = 3$	$10^{-4}$	20,136	1,481,708	0.01	-7258.5%
5	Beale	$l = 2, m = 3$	$10^{-4}$	97,975	120,000	0.82	-22.48%
6	Jenrich and Sampson	$l = 2, m = 10$	$10^{-4}$	9,168,633	9,141,258	1.00	0.3%
7	Helical valley	$l = 3, m = 3$	$10^{-4}$	107,243	108,134	0.99	-83.08%
8	Bard	$l = 3, m = 15$	$10^{-4}$	77,256,257	97,838,159	0.79	-26.64%
9	Gaussian	$l = 3, m = 15$	$10^{-4}$	1,488,304	8,961,152	0.16	-502.1%
10	Meyer	$l = 3, m = 16$	$10^{-2}$	156,851,462	155,613,851	1.00	1.48%
11	Gulf research	$l = 3, m = 3$	$10^{-4}$	$> 518,626,071$	$5.55 \times 10^9$	*	*
12	Box three dimensional	$l = 3, m = 3$	$10^{-4}$	27,016,779	42,425,722	0.64	-57.03%
13	Powell singular	$l = 4, m = 4$	$10^{-4}$	1,614,009	1,474,760	1.09	8.625%
14	Wood	$l = 4, m = 6$	$10^{-4}$	124,126	690,722	0.18	-456.47%
15	Kowalik and Osborne	$l = 4, m = 11$	$10^{-4}$	364,840,577	759,876,285	0.48	-108.28%
16	Brown and Dennis	$l = 6, m = 20$	$10^{-4}$	103,925,442	104,551,931	0.99	-60.28%
17	Osborne 1	$l = 5, m = 33$	$10^{-4}$	265,719,213	294,136,090	0.90	-10.69%
18	Biggs EXP 6	$l = 6, m = 13$	$10^{-2}$	570,154,433	$1.73 \times 10^9$	0.33	-203.53%
19	Osborne 2	$l = 11, m = 65$	$10^{-2}$	$4.94 \times 10^9$	$2 \times 10^9$	2.47	59.46%
20	Watson	$l = 6, m = 31$	$10^{-2}$	214,144,957	310,062,391	0.69	-44.79%
21	Extended Rosenbrock	$l = 4, m = 4$	$10^{-4}$	50,120	319,404	0.16	-537.28%
22	Extended Powell singular	$l = 4, m = 4$	$10^{-4}$	793,703	738,140	1.08	7%

Table 4.8 (Contd.)

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
23	Penalty I	$l = 4, m = 8$	$10^{-4}$	10,281,240	48,177,871	0.21	-368.59%
24	Penalty II	$l = 4, m = 8$	$10^{-4}$	5,595,454	430,032,828	0.01	-7585.4%
25	Variably dim.	$l = 2, m = 4$	$10^{-4}$	7,804	13,863	0.56	-77.64%
26	Trigonometric	$l = 4, m = 4$	$10^{-4}$	3,432,098	3,380,207	1.01	1.51%
27	Brown almost linear	$l = 4, m = 4$	$10^{-4}$	32,234,685	42,286,626	0.76	-31.18%
28	Discrete boundary value	$l = 2, m = 2$	$10^{-4}$	17,142	21,782	0.79	-27.07%
29	Discrete integral eq.	$l = 4, m = 4$	$10^{-4}$	85,535	212,077	0.4	-147.94%
30	Broyden tridiagonal	$l = 4, m = 4$	$10^{-4}$	209,003	847,888	0.25	-305.68%
31	Broyden branded	$l = 4, m = 4$	$10^{-4}$	211,585	431,411	0.49	-103.89%
32	Linear full rank	$l = 4, m = 4$	$10^{-4}$	337,027	1,027,903	0.33	-204.99%
33	Linear rank-1	$l = 3, m = 4$	$10^{-2}$	>40,411,869	211,236,148	*	*
34	Linear rank-1 with zero column and rows	$l = 2, m = 4$	$10^{-4}$	>46,899,186	271,139,362	*	*
35	Chebyquad	$l = 2, m = 2$	$10^{-4}$	24,001	25,524	0.94	-6.35%
36	SHCB: Six hump camel	$l = 2, m = 1$	$10^{-4}$	313,131	239,814	0.77	-30.57%
37	THCB: Three hump camel	$l = 2, m = 1$	$10^{-4}$	133,535	104,910	1.27	21.44%
38	BR: Branin	$l = 2, m = 1$	$10^{-4}$	41,013	63,648	0.64	-55.19%
39	L3: Levy	$l = 2, m = 1$	$10^{-4}$	3,888,033	6,648,010	0.58	-70.99%
40	L5: Levy	$l = 2, m = 1$	$10^{-4}$	829,501	3,361,593	0.25	-305.25%
41	L8: Levy	$l = 3, m = 1$	$10^{-4}$	20,358	1,077,697	0.02	-5193.7%
42	L18: Levy	$l = 7, m = 1$	$10^{-4}$	424,939	1,493,004	0.28	-251.34%
43	Schw2.5: Booth	$l = 2, m = 1$	$10^{-4}$	14,892	17,780	0.84	-19.39%
44	Schw2.18: Matyas	$l = 2, m = 1$	$10^{-4}$	85,962	78,546	1.09	8.63%
45	Schw3.1: Schwefel	$l = 3, m = 1$	$10^{-4}$	15,298	52,498	0.29	-243.17%
46	Schw3.1p: Schwefel	$l = 3, m = 1$	$10^{-4}$	108,450	387,136	0.28	-256.97%
47	Schw3.2: Schwefel	$l = 3, m = 1$	$10^{-4}$	26,866	49,604	0.54	-84.63%
48	Griew5: Griewank	$l = 5, m = 1$	$10^{-4}$	4,713,853	16,471,942	0.29	-249.44%
49	GP: Goldstein and price	$l = 2, m = 1$	$10^{-4}$	82,163,581	74,830,648	1.09	8.92%
50	R4: Ratz	$l = 2, m = 1$	$10^{-4}$	681,806	797,686	0.85	-16.99%

TABLE 4.9. Computational time for basic and proposed algorithms with termination criterion A.

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1	Rosenbrock	$l = 2, m = 2$	$10^{-4}$	0.98	0.45	2.18	54.08%
2	Freudenstein and Roth	$l = 2, m = 2$	$10^{-4}$	13.88	0.8	17.35	94.24%
3	Powell badly scaled	$l = 2, m = 2$	$10^{-4}$	$> 36,000$	$1.18 \times 10^4$	*	*
4	Brown badly scaled	$l = 2, m = 3$	$10^{-4}$	1.25	1.75	0.71	-40%
5	Beale	$l = 2, m = 3$	$10^{-4}$	4.8	0.59	8.14	87.71%
6	Jenrich and Sampson	$l = 2, m = 10$	$10^{-4}$	37.47	2.61	14.36	93.03%
7	Helical valley	$l = 3, m = 3$	$10^{-4}$	4.88	1.1	4.44	77.46%
8	Bard	$l = 3, m = 15$	$10^{-4}$	$1.3 \times 10^3$	30.1	43.19	97.68%
9	Gaussian	$l = 3, m = 15$	$10^{-4}$	4.28	2.5	1.71	41.59%
10	Meyer	$l = 3, m = 16$	$10^{-2}$	768.32	47.14	16.29	93.86%
11	Gulf research	$l = 3, m = 3$	$10^{-4}$	$> 36,000$	$1.93 \times 10^4$	*	*
12	Box three dim.	$l = 3, m = 3$	$10^{-4}$	731.57	11.04	66.27	98.49%
13	Powell singular	$l = 4, m = 4$	$10^{-4}$	74.7	1.86	40.16	97.51%
14	Wood	$l = 4, m = 6$	$10^{-4}$	5.07	1.92	2.64	62.13%
15	Kowalik and Osborne	$l = 4, m = 11$	$10^{-4}$	$2.13 \times 10^4$	229.64	92.75	98.92%
16	Brown and Dennis	$l = 6, m = 20$	$10^{-4}$	438.38	24.29	18.05	94.46%
17	Osborne 1	$l = 5, m = 33$	$10^{-4}$	349.17	77.37	4.51	77.84%
18	Biggs EXP 6	$l = 6, m = 13$	$10^{-2}$	$7.42 \times 10^3$	508.38	14.59	93.15%
19	Osborne 2	$l = 11, m = 65$	$10^{-2}$	$5.96 \times 10^3$	601.35	9.91	89.91%
20	Watson	$l = 6, m = 31$	$10^{-2}$	$6.37 \times 10^2$	106.84	5.96	98.93%
21	Extended Rosenbrock	$l = 4, m = 4$	$10^{-4}$	2.19	1.3	1.68	40.64%
22	Extended Powell singular	$l = 4, m = 4$	$10^{-4}$	34.85	1.56	22.34	95.52%

Table 4.9 (Contd.)

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
23	Penalty I	$l = 4, m = 5$	$10^{-4}$	$9.22 \times 10^2$	24.78	37.21	97.31%
24	Penalty II	$l = 4, m = 8$	$10^{-4}$	$1.1 \times 10^2$	139.7	0.79	-27%
25	Variably dim.	$l = 2, m = 4$	$10^{-4}$	0.43	0.39	1.1	9.3%
26	Trigonometric	$l = 4, m = 4$	$10^{-4}$	54.66	5.47	9.99	89.99%
27	Brown almost linear	$l = 4, m = 4$	$10^{-4}$	$2.59 \times 10^3$	15.93	162.59	99.38%
28	Discrete boundary value	$l = 2, m = 2$	$10^{-4}$	1.02	0.4	2.55	60.78%
29	Discrete integral eq.	$l = 4, m = 4$	$10^{-4}$	3.16	2.6	1.21	17.72%
30	Broyden tridiagonal	$l = 4, m = 4$	$10^{-4}$	8.92	1.66	5.37	81.39%
31	Broyden branded	$l = 4, m = 4$	$10^{-4}$	7.98	1.91	4.18	76.07%
32	Linear full rank	$l = 4, m = 4$	$10^{-4}$	15.09	1.33	11.19	91.19%
33	Linear rank-1	$l = 3, m = 4$	$10^{-2}$	>36000	104.03	*	*
34	Linear rank-1 with zero column and rows	$l = 2, m = 4$	$10^{-4}$	>36000	153.83	*	*
35	Chebyquad	$l = 2, m = 2$	$10^{-4}$	1.73	0.35	4.94	79.77%
36	SHCB: Six hump camel	$l = 2, m = 1$	$10^{-4}$	16.45	1.05	15.67	93.62%
37	THCB: Three hump camel	$l = 2, m = 1$	$10^{-4}$	8.01	0.6	13.35	92.51%
38	BR: Branin	$l = 2, m = 1$	$10^{-4}$	1.35	0.59	2.29	56.29%
39	L3: Levy	$l = 2, m = 1$	$10^{-4}$	72.59	7.43	9.77	89.76%
40	L5: Levy	$l = 2, m = 1$	$10^{-4}$	16.32	6.04	2.7	62.99%
41	L8: Levy	$l = 3, m = 1$	$10^{-4}$	2.62	2.62	1	0.00%
42	L18: Levy	$l = 7, m = 1$	$10^{-4}$	8.58	9.59	0.89	-11.77%
43	Schw2.5: Booth	$l = 2, m = 1$	$10^{-4}$	1.22	0.42	2.9	65.57%
44	Schw2.18: Matyas	$l = 2, m = 1$	$10^{-4}$	8.14	0.52	15.65	93.61%
45	Schw3.1: Schwefel	$l = 3, m = 1$	$10^{-4}$	0.97	0.95	1.02	2.06%
46	Schw3.1p: Schwefel	$l = 3, m = 1$	$10^{-4}$	4.59	2	2.29	56.43%
47	Schw3.2: Schwefel	$l = 3, m = 1$	$10^{-4}$	1.82	0.66	2.76	63.74%
48	Griew5: Griewank	$l = 5, m = 1$	$10^{-4}$	98.24	18.21	5.39	81.46%
49	GP: Goldstein and price	$l = 2, m = 1$	$10^{-4}$	$1.08 \times 10^4$	22.42	481.71	99.79%
50	R4: Ratz	$l = 2, m = 1$	$10^{-4}$	17.78	1.3	13.68	92.69%

TABLE 4.10. Maximum list length for basic and proposed algorithms with termination criterion A.

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1	Rosenbrock	$l = 2, m = 2$	$10^{-4}$	11	16	0.69	-45.45%
2	Freudenstein and Roth	$l = 2, m = 2$	$10^{-4}$	75	66	1.14	12%
3	Powell badly scaled	$l = 2, m = 2$	$10^{-4}$	$> 45,018$	618,568	*	*
4	Brown badly scaled	$l = 2, m = 3$	$10^{-4}$	19	1190	0.02	-6163.2%
5	Beale	$l = 2, m = 3$	$10^{-4}$	27	23	1.17	14.81%
6	Jenrich and Sampson	$l = 2, m = 10$	$10^{-4}$	70	46	1.52	34.29%
7	Helical valley	$l = 3, m = 3$	$10^{-4}$	12	10	1.2	20%
8	Bard	$l = 3, m = 15$	$10^{-4}$	5552	4805	1.16	13.45%
9	Gaussian	$l = 3, m = 15$	$10^{-4}$	50	72	0.69	-44%
10	Meyer	$l = 3, m = 16$	$10^{-2}$	5115	5114	1.00	0.02%
11	Gulf research	$l = 3, m = 3$	$10^{-4}$	$> 56,867$	933,342	*	*
12	Box three dimensional	$l = 3, m = 3$	$10^{-4}$	2843	2881	0.99	-1.34%
13	Powell singular	$l = 4, m = 4$	$10^{-4}$	72	72	1.00	0.00%
14	Wood	$l = 4, m = 6$	$10^{-4}$	22	51	0.43	-131.82%
15	Kowalik and Osborne	$l = 4, m = 11$	$10^{-4}$	30,593	57,060	0.54	-86.51%
16	Brown and Dennis	$l = 4, m = 20$	$10^{-4}$	322	207	1.56	35.71%
17	Osborne 1	$l = 5, m = 33$	$10^{-4}$	1,082	936	1.17	13.49%
18	Biggs EXP 6	$l = 6, m = 13$	$10^{-2}$	20,126	47,482	0.42	-135.92%
19	Osborne 2	$l = 11, m = 65$	$10^{-2}$	2,343	544	4.31	76.78%
20	Watson	$l = 6, m = 31$	$10^{-2}$	1,432	1,099	1.3	23.25%
21	Extended Rosenbrock	$l = 4, m = 4$	$10^{-4}$	11	22	0.5	-100%
22	Extended Powell singular	$l = 4, m = 4$	$10^{-4}$	33	32	1.03	3.03%

Table 4.10 (Contd.)

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
23	Penalty I	$l = 4, m = 5$	$10^{-4}$	6,876	21,333	0.32	-210.25%
24	Penalty II	$l = 4, m = 8$	$10^{-4}$	812	26,919	0.03	$-3.22 \times 10^3\%$
25	Variably dim.	$l = 2, m = 4$	$10^{-4}$	2	2	0.00	0.00%
26	Trigonometric	$l = 4, m = 4$	$10^{-4}$	31	27	1.15	12.9%
27	Brown almost linear	$l = 4, m = 4$	$10^{-4}$	5,779	3,121	1.85	45.99%
28	Discrete boundary value	$l = 2, m = 2$	$10^{-4}$	8	6	1.33	25%
29	Discrete integral eq.	$l = 4, m = 4$	$10^{-4}$	7	11	0.64	-57.14%
30	Broyden tridiagonal	$l = 4, m = 4$	$10^{-4}$	85	107	0.79	-25.88%
31	Broyden branded	$l = 4, m = 4$	$10^{-4}$	56	35	1.6	37.5%
32	Linear full rank	$l = 4, m = 4$	$10^{-4}$	105	59	1.78	43.81%
33	Linear rank-1	$l = 3, m = 4$	$10^{-2}$	>56,360	290,704	*	*
34	Linear rank-1 with zero column and rows	$l = 2, m = 4$	$10^{-4}$	>48,419	349,525	*	*
35	Chebyquad	$l = 2, m = 2$	$10^{-4}$	16	12	1.33	25%
36	SHCB: Six hump camel	$l = 2, m = 1$	$10^{-4}$	132	88	1.5	33.33%
37	THCB: Three hump camel	$l = 2, m = 1$	$10^{-4}$	54	42	1.29	22.22%
38	BR: Branin	$l = 2, m = 1$	$10^{-4}$	7	7	1.00	0.00%
39	L3: Levy	$l = 2, m = 1$	$10^{-4}$	158	259	0.61	-63.92%
40	L5: Levy	$l = 2, m = 1$	$10^{-4}$	36	169	0.21	-369.44%
41	L8: Levy	$l = 3, m = 1$	$10^{-4}$	8	26	0.31	-225%
42	L18: Levy	$l = 7, m = 1$	$10^{-4}$	7	18	0.39	-157.14%
43	Schw2.5: Booth	$l = 2, m = 1$	$10^{-4}$	6	7	0.86	-16.67%
44	Schw2.18: Matyas	$l = 2, m = 1$	$10^{-4}$	24	28	0.86	-16.67%
45	Schw3.1: Schwefel	$l = 3, m = 1$	$10^{-4}$	3	7	0.43	-133.33%
46	Schw3.1p: Schwefel	$l = 3, m = 1$	$10^{-4}$	31	139	0.22	-348.39%
47	Schw3.2: Schwefel	$l = 3, m = 1$	$10^{-4}$	12	12	1.00	0.00%
48	Griew5: Griewank	$l = 5, m = 1$	$10^{-4}$	32	32	1.00	0.00%
49	GP: Goldstein and price	$l = 2, m = 1$	$10^{-4}$	5,580	8,101	0.69	-45.18%
50	R4: Ratz	$l = 2, m = 1$	$10^{-4}$	92	76	1.21	17.39%

TABLE 4.11. Function evaluations for basic and proposed algorithms with termination criterion B.

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1	Rosenbrock	$l = 2, m = 2$	$10^{-4}$	150	592	0.25	-294.67%
2	Freudenstein and Roth	$l = 2, m = 2$	$10^{-4}$	1,926	2,244	0.86	-16.51%
3	Powell badly scaled	$l = 2, m = 2$	$10^{-4}$	$> 244,440$	1,122,986	*	*
4	Brown badly scaled	$l = 2, m = 3$	$10^{-4}$	$> 2,918,744$	104,366	*	*
5	Beale	$l = 2, m = 3$	$10^{-4}$	756	648	1.17	14.29%
6	Jenrich and Sampson	$l = 2, m = 10$	$10^{-4}$	1,338	1,342	0.99	-0.29%
7	Helical valley	$l = 3, m = 3$	$10^{-4}$	356	386	0.92	-8.43%
8	Bard	$l = 3, m = 15$	$10^{-4}$	613,786	79,648	7.71	87.02%
9	Gaussian	$l = 3, m = 16$	$10^{-4}$	504	1,556	0.32	-208.73%
10	Meyer	$l = 3, m = 15$	$10^{-2}$	$> 534,367$	48,290	*	*
11	Gulf research	$l = 3, m = 3$	$10^{-4}$	$> 416,557$	327,062	*	*
12	Box three dimensional	$l = 3, m = 3$	$10^{-4}$	290,421	83,179	3.49	71.36%
13	Powell singular	$l = 4, m = 4$	$10^{-4}$	5,462	5,462	1.00	0.00%
14	Wood	$l = 4, m = 6$	$10^{-4}$	400	1,988	0.2	-397.00%
15	Kowalik and Osborne	$l = 4, m = 11$	$10^{-4}$	$> 193,814$	333,996	*	*
16	Brown and Dennis	$l = 4, m = 20$	$10^{-4}$	173,221	18,550	9.34	89.29%
17	Osborne 1	$l = 5, m = 33$	$10^{-4}$	39,542	27,359	1.45	30.82%
18	Biggs EXP 6	$l = 6, m = 13$	$10^{-2}$	79,295	37,993	2.09	52.08%
19	Osborne 2	$l = 11, m = 65$	$10^{-2}$	102,810	17,661	5.82	82.82%
20	Watson	$l = 6, m = 31$	$10^{-2}$	37,731	38,459	0.98	-1.93%
21	Extended Rosenbrock	$l = 4, m = 4$	$10^{-4}$	292	1,372	0.19	-423.66%
22	Extended Powell singular	$l = 4, m = 4$	$10^{-4}$	2672	2674	0.99	-0.08%



Table 4.11 (Contd.)

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
23	Penalty I	$l = 4, m = 5$	$10^{-4}$	$>187,224$	184,876	*	*
24	Penalty II	$l = 4, m = 8$	$10^{-4}$	$>192,884$	282,360	*	*
25	Variably dim.	$l = 2, m = 4$	$10^{-4}$	42	74	0.57	-76.19%
26	Trigonometric	$l = 4, m = 4$	$10^{-4}$	882	882	1.00	0.00%
27	Brown almost linear	$l = 4, m = 4$	$10^{-4}$	266,612	125,362	2.13	52.98%
28	Discrete boundary value	$l = 2, m = 2$	$10^{-4}$	146	126	1.16	13.69%
29	Discrete integral eq.	$l = 4, m = 4$	$10^{-4}$	132	234	0.56	-77.30%
30	Broyden tridiagonal	$l = 4, m = 4$	$10^{-4}$	794	2,536	0.31	-219.39%
31	Broyden branded	$l = 4, m = 4$	$10^{-4}$	396	814	0.49	-105.56%
32	Linear full rank	$l = 4, m = 4$	$10^{-4}$	1,280	3,712	0.34	-190.00%
33	Linear rank-1	$l = 3, m = 4$	$10^{-2}$	134,888	34,030	3.96	74.77%
34	Linear rank-1 with zero column and rows	$l = 2, m = 4$	$10^{-4}$	196,570	125,362	1.56	36.23%
35	Chebyquad	$l = 2, m = 2$	$10^{-4}$	$>253,870$	62,988	*	*
36	SHCB: Six hump camel	$l = 2, m = 1$	$10^{-4}$	1,446	1,378	1.05	4.71%
37	THCB: Three hump camel	$l = 2, m = 1$	$10^{-4}$	886	810	1.09	8.58%
38	BR: Branin	$l = 2, m = 1$	$10^{-4}$	122	184	0.66	-50.82%
39	L3: Levy	$l = 2, m = 1$	$10^{-4}$	1,430	2,696	0.53	-88.53%
40	L5: Levy	$l = 2, m = 1$	$10^{-4}$	358	1,510	0.24	-321.79%
41	L8: Levy	$l = 3, m = 1$	$10^{-4}$	62	240	0.26	-287.01%
42	L18: Levy	$l = 7, m = 1$	$10^{-4}$	184	644	0.29	-250%
43	Schw2.5: Booth	$l = 2, m = 1$	$10^{-4}$	150	206	0.73	-37.33%
44	Schw2.18: Matyas	$l = 2, m = 1$	$10^{-4}$	$>36000$	1,326	*	*
45	Schw3.1: Schwefel	$l = 3, m = 1$	$10^{-4}$	78	296	0.26	-279.49%
46	Schw3.1p: Schwefel	$l = 3, m = 1$	$10^{-4}$	89,106	89,310	0.99	-231%
47	Schw3.2: Schwefel	$l = 3, m = 1$	$10^{-4}$	224	444	0.5	-98.21%
48	Griew5: Griewank	$l = 5, m = 1$	$10^{-4}$	$>116400$	5,182	*	*
49	GP: Goldstein and price	$l = 2, m = 1$	$10^{-4}$	88,020	86,628	1.02	1.58%
50	R4: Ratz	$l = 2, m = 1$	$10^{-4}$	1,294	1,574	0.82	-21.64%

TABLE 4.12. *Flops* for basic and proposed algorithms with termination criterion B.

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1	Rosenbrock	$l = 2, m = 2$	$10^{-4}$	16,027	53,367	0.3	-232.98%
2	Freudenstein and Roth	$l = 2, m = 2$	$10^{-4}$	450,068	348,920	1.28	-22.47%
3	Powell badly scaled	$l = 2, m = 2$	$10^{-4}$	$> 8.53 \times 10^9$	353,189,839	*	*
4	Brown badly scaled	$l = 2, m = 3$	$10^{-4}$	$> 437,810,978$	11,725,776	*	*
5	Beale	$l = 2, m = 3$	$10^{-4}$	181,496	130,095	1.39	28.32%
6	Jenrich and Sampson	$l = 2, m = 10$	$10^{-4}$	3,754,786	3,675,485	1.02	2.1%
7	Helical valley	$l = 3, m = 3$	$10^{-4}$	113,250	114,002	0.99	-0.66%
8	Bard	$l = 3, m = 15$	$10^{-4}$	$1.13 \times 10^{10}$	111,450,644	101.27	99.01%
9	Gaussian	$l = 3, m = 15$	$10^{-4}$	2,956,930	8,963,792	0.33	-203.15%
10	Meyer	$l = 3, m = 16$	$10^{-2}$	$> 1.11 \times 10^{10}$	145,900,198	*	*
11	Gulf research	$l = 3, m = 3$	$10^{-4}$	$> 9.87 \times 10^9$	523,567,828	*	*
12	Box three dimensional	$l = 3, m = 3$	$10^{-4}$	67,365,662	42,526,087	1.58	36.87%
13	Powell singular	$l = 4, m = 4$	$10^{-4}$	2,470,078	1,556,090	1.58	37.00%
14	Wood	$l = 4, m = 6$	$10^{-4}$	169,457	726,677	0.23	-328.83%
15	Kowalik and Osborne	$l = 4, m = 11$	$10^{-4}$	$> 1.01 \times 10^9$	760,996,038	*	*
16	Brown and Dennis	$l = 6, m = 20$	$10^{-4}$	58,184,027	52,903,173	1.09	9.07%
17	Osborne 1	$l = 5, m = 33$	$10^{-4}$	299,422,685	279,891,059	1.07	6.81%
18	Biggs EXP 6	$l = 6, m = 13$	$10^{-2}$	485,637,919	228,041,529	2.13	53.04%
19	Osborne 2	$l = 11, m = 65$	$10^{-2}$	$4.58 \times 10^9$	$1.09 \times 10^9$	4.17	76.02%
20	Watson	$l = 6, m = 31$	$10^{-2}$	271,541,024	298,918,263	0.91	-10.08%
21	Extended Rosenbrock	$l = 4, m = 4$	$10^{-4}$	71,022	337,294	0.21	-374.91%
22	Extended Powell singular	$l = 4, m = 4$	$10^{-4}$	976,677	761,642	1.28	22.02%

Table 4.12 (Contd.)

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
23	Penalty I	$l = 4, m = 8$	$10^{-4}$	$>1.02 \times 10^9$	632,126,399	*	*
24	Penalty II	$l = 4, m = 8$	$10^{-4}$	$>9.62 \times 10^9$	451,369,117	*	*
25	Variably dim.	$l = 2, m = 4$	$10^{-4}$	8,273	14,370	0.58	-73.69%
26	Trigonometric	$l = 4, m = 4$	$10^{-4}$	3,462,401	3,381,955	1.02	2.32%
27	Brown almost linear	$l = 4, m = 4$	$10^{-4}$	$2.45 \times 10^9$	46,190,896	53.04	98.11%
28	Discrete boundary value	$l = 2, m = 2$	$10^{-4}$	30,818	23,414	1.32	24.02%
29	Discrete integral eq.	$l = 4, m = 4$	$10^{-4}$	141,014	217,454	0.65	-54.21%
30	Broyden tridiagonal	$l = 4, m = 4$	$10^{-4}$	348,179	868,626	0.40	-149.47%
31	Broyden branded	$l = 4, m = 4$	$10^{-4}$	231,244	432,606	0.53	-87.07%
32	Linear full rank	$l = 4, m = 4$	$10^{-4}$	639,859	1,120,696	0.57	-75.15%
33	Linear rank-1	$l = 3, m = 4$	$10^{-2}$	$7.96 \times 10^9$	6,444,498	1235.32	99.92%
34	Linear rank-1 with zero column and rows	$l = 2, m = 4$	$10^{-4}$	$6.2 \times 10^9$	49,201,583	126.08	99.21%
35	Chebysquad	$l = 2, m = 2$	$10^{-4}$	$>7.66 \times 10^9$	$2.31 \times 10^9$	*	*
36	SHCB: Six hump camel	$l = 2, m = 1$	$10^{-4}$	460,278	229,692	2	50.1%
37	THCB: Three hump camel	$l = 2, m = 1$	$10^{-4}$	187,441	106,110	1.77	43.39%
38	BR: Branin	$l = 2, m = 1$	$10^{-4}$	46,223	63,920	0.72	-38.29%
39	L3: Levy	$l = 2, m = 1$	$10^{-4}$	3,258,434	5,897,860	0.55	-81.00%
40	L5: Levy	$l = 2, m = 1$	$10^{-4}$	751,583	3,275,496	0.23	-335.81%
41	L8: Levy	$l = 3, m = 1$	$10^{-4}$	203,805	1,078,146	0.19	-429.01%
42	L18: Levy	$l = 7, m = 1$	$10^{-4}$	426,743	1,495,488	0.29	-250.04%
43	Schw2.5: Booth	$l = 2, m = 1$	$10^{-4}$	16,259	18,627	0.87	-14.56%
44	Schw2.18: Matyas	$l = 2, m = 1$	$10^{-4}$	$>$	80,586	*	*
45	Schw3.1: Schwefel	$l = 3, m = 1$	$10^{-4}$	15,607	53,051	0.29	-239.92%
46	Schw3.1p: Schwefel	$l = 3, m = 1$	$10^{-4}$	$3.63 \times 10^9$	$4.38 \times 10^7$	82.75	98.79%
47	Schw3.2: Schwefel	$l = 3, m = 1$	$10^{-4}$	35,433	55,477	0.64	-56.57%
48	Griew5: Griewank	$l = 5, m = 1$	$10^{-4}$	$>8.54 \times 10^9$	16,485,718	*	*
49	GP: Goldstein and price	$l = 2, m = 1$	$10^{-4}$	$5.92 \times 10^8$	$7.35 \times 10^7$	8.07	87.61%
50	R4: Ratz	$l = 2, m = 1$	$10^{-4}$	770,310	800,060	0.96	-3.86%

TABLE 4.13. Computational time for basic and proposed algorithms with termination criterion B.

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1	Rosenbrock	$l = 2, m = 2$	$10^{-4}$	1.15	0.45	2.55	60.87%
2	Freudenstein and Roth	$l = 2, m = 2$	$10^{-4}$	18.90	0.8	23.63	95.24%
3	Powell badly scaled	$l = 2, m = 2$	$10^{-4}$	$> 36,000$	147.04	*	*
4	Brown badly scaled	$l = 2, m = 3$	$10^{-4}$	$> 4.38 \times 10^8$	11,725,776	*	*
5	Beale	$l = 2, m = 3$	$10^{-4}$	7.5	0.57	13.16	92.40%
6	Jenrich and Sampson	$l = 2, m = 10$	$10^{-4}$	13.90	1.2	11.58	91.37%
7	Helical valley	$l = 3, m = 3$	$10^{-4}$	4.9	1.1	4.45	77.55%
8	Bard	$l = 3, m = 15$	$10^{-4}$	35,983	37.58	957.50	99.89%
9	Gaussian	$l = 3, m = 15$	$10^{-4}$	8.22	2.5	3.28	69.59%
10	Meyer	$l = 3, m = 16$	$10^{-2}$	$> 360000$	48.22	*	*
11	Gulf research	$l = 3, m = 3$	$10^{-4}$	$> 36,000$	170.6	*	*
12	Box three dimensional	$l = 3, m = 3$	$10^{-4}$	3600.81	11.36	316.97	99.68%
13	Powell singular	$l = 4, m = 4$	$10^{-4}$	75.2	1.90	39.58	97.47%
14	Wood	$l = 4, m = 6$	$10^{-4}$	6.10	2.00	3.05	67.21%
15	Kowalik and Osborne	$l = 4, m = 11$	$10^{-4}$	$> 360000$	307.33	*	*
16	Brown and Dennis	$l = 6, m = 20$	$10^{-4}$	231.91	13.20	17.57	94.31%
17	Osborne 1	$l = 5, m = 33$	$10^{-4}$	386.50	77.19	5.00	80.03%
18	Biggs EXP 6	$l = 6, m = 13$	$10^{-2}$	1197.90	64.74	18.50	94.60%
19	Osborne 2	$l = 11, m = 65$	$10^{-2}$	3490	322.60	10.82	90.76%
20	Watson	$l = 6, m = 31$	$10^{-2}$	648.5	106.82	6.07	83.53%
21	Extended Rosenbrock	$l = 4, m = 4$	$10^{-4}$	2.9	1.3	2.23	55.17%
22	Extended Powell singular	$l = 4, m = 4$	$10^{-4}$	33.60	1.50	22.40	95.54%

Table 4.13 (Contd.)

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
23	Penalty I	$l = 4, m = 5$	$10^{-4}$	>360000	35.53	*	*
24	Penalty II	$l = 4, m = 8$	$10^{-4}$	>360000	156.76	*	*
25	Variably dim.	$l = 2, m = 4$	$10^{-4}$	0.43	0.41	1.08	4.65%
26	Trigonometric	$l = 4, m = 4$	$10^{-4}$	52.10	5.10	10.22	90.21%
27	Brown almost linear	$l = 4, m = 4$	$10^{-4}$	10,104	18.50	546.16	99.82%
28	Discrete boundary value	$l = 2, m = 2$	$10^{-4}$	1.58	0.41	3.85	74.05%
29	Discrete integral eq.	$l = 4, m = 4$	$10^{-4}$	4.6	2.6	1.77	43.48%
30	Broyden tridiagonal	$l = 4, m = 4$	$10^{-4}$	10.7	1.60	6.69	85.05%
31	Broyden branded	$l = 4, m = 4$	$10^{-4}$	7.30	1.80	4.05	75.34%
32	Linear full rank	$l = 4, m = 4$	$10^{-4}$	16.10	1.40	11.50	91.30%
33	Linear rank-1	$l = 3, m = 4$	$10^{-2}$	$2.61 \times 10^4$	2.99	8729.10	99.99%
34	Linear rank-1 with zero column and rows	$l = 2, m = 4$	$10^{-4}$	$2.59 \times 10^4$	28.58	909.38	99.89
35	Chebysquad	$l = 2, m = 2$	$10^{-4}$	>360000	$2.31 \times 10^9$	*	*
36	SHCB: Six hump camel	$l = 2, m = 1$	$10^{-4}$	16.31	0.88	18.53	94.60%
37	THCB: Three hump camel	$l = 2, m = 1$	$10^{-4}$	9.11	0.59	15.44	93.52%
38	BR: Branin	$l = 2, m = 1$	$10^{-4}$	1.48	0.65	2.27	56.08%
39	L3: Levy	$l = 2, m = 1$	$10^{-4}$	61.52	5.23	11.76	91.5%
40	L5: Levy	$l = 2, m = 1$	$10^{-4}$	14.7	4.41	3.33	70.00%
41	L8: Levy	$l = 3, m = 1$	$10^{-4}$	2.16	2.66	0.812	-23.15%
42	L18: Levy	$l = 7, m = 1$	$10^{-4}$	8.6	9.41	0.91	-9.42%
43	Schw2.5: Booth	$l = 2, m = 1$	$10^{-4}$	1.22	0.41	2.98	66.39%
44	Schw2.18: Matyas	$l = 2, m = 1$	$10^{-4}$	>360000	0.49	*	*
45	Schw3.1: Schwefel	$l = 3, m = 1$	$10^{-4}$	0.9	0.94	0.96	-4.44%
46	Schw3.1p: Schwefel	$l = 3, m = 1$	$10^{-4}$	$1.43 \times 10^4$	18.03	794.01	99.87%
47	Schw3.2: Schwefel	$l = 3, m = 1$	$10^{-4}$	2.22	0.68	3.26	69.37%
48	Griew5: Griewank	$l = 5, m = 1$	$10^{-4}$	>360000	17.42	*	*
49	GP: Goldstein and price	$l = 2, m = 1$	$10^{-4}$	$3.91 \times 10^3$	2,068	1.89	47.11%
50	R4: Ratz	$l = 2, m = 1$	$10^{-4}$	17.98	1.27	14.16	92.94%

TABLE 4.14. Maximum list length for basic and proposed algorithms with termination criterion B.

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1	Rosenbrock	$l = 2, m = 2$	$10^{-4}$	11	16	0.69	-45.45%
2	Freudenstein and Roth	$l = 2, m = 2$	$10^{-4}$	75	66	1.14	12%
3	Powell badly scaled	$l = 2, m = 2$	$10^{-4}$	$> 45,018$	618,568	*	*
4	Brown badly scaled	$l = 2, m = 3$	$10^{-4}$	$> 16$	16,385	*	*
5	Beale	$l = 2, m = 3$	$10^{-4}$	27	23	1.17	14.81%
6	Jenrich and Sampson	$l = 2, m = 10$	$10^{-4}$	70	46	1.52	34.29%
7	Helical valley	$l = 3, m = 3$	$10^{-4}$	12	10	1.2	16.67%
8	Bard	$l = 3, m = 15$	$10^{-4}$	12,714	5781	2.19	54.53%
9	Gaussian	$l = 3, m = 15$	$10^{-4}$	54	72	0.75	-33.33%
10	Meyer	$l = 3, m = 16$	$10^{-2}$	$> 4612$	9745	*	*
11	Gulf research	$l = 3, m = 3$	$10^{-4}$	$> 59,825$	44,875	*	*
12	Box three dimensional	$l = 3, m = 3$	$10^{-4}$	4011	2881	1.39	28.17%
13	Powell singular	$l = 4, m = 4$	$10^{-4}$	72	72	1.00	0.00%
14	Wood	$l = 4, m = 6$	$10^{-4}$	22	51	0.43	-131.82%
15	Kowalik and Osborne	$l = 4, m = 11$	$10^{-4}$	$> 37,133$	57,060	*	*
16	Brown and Dennis	$l = 4, m = 20$	$10^{-4}$	322	207	1.56	35.71%
17	Osborne 1	$l = 5, m = 33$	$10^{-4}$	1,082	936	1.17	13.49%
18	Biggs EXP 6	$l = 6, m = 13$	$10^{-2}$	2585	1698	1.52	34.31%
19	Osborne 2	$l = 11, m = 65$	$10^{-2}$	2,343	544	4.31	76.78%
20	Watson	$l = 6, m = 31$	$10^{-2}$	1,405	1,099	1.28	21.78%
21	Extended Rosenbrock	$l = 4, m = 4$	$10^{-4}$	11	22	0.5	-100%
22	Extended Powell singular	$l = 4, m = 4$	$10^{-4}$	33	32	1.03	3.03%

Table 4.14 (Contd.)

S.No	Test function	Dimension	$\varepsilon_F$	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
23	Penalty I	$l = 4, m = 5$	$10^{-4}$	>44,742	27,065	*	*
24	Penalty II	$l = 4, m = 8$	$10^{-4}$	>40,705	26,919	*	*
25	Variably dim.	$l = 2, m = 4$	$10^{-4}$	2	2	0.00	0.00%
26	Trigonometric	$l = 4, m = 4$	$10^{-4}$	31	27	1.15	12.9%
27	Brown almost linear	$l = 4, m = 4$	$10^{-4}$	5,962	3,121	1.91	47.65%
28	Discrete boundary value	$l = 2, m = 2$	$10^{-4}$	9	6	1.50	33.33%
29	Discrete integral eq.	$l = 4, m = 4$	$10^{-4}$	8	11	0.73	-37.50%
30	Broyden tridiagonal	$l = 4, m = 4$	$10^{-4}$	85	107	0.79	-25.88%
31	Broyden branded	$l = 4, m = 4$	$10^{-4}$	56	35	1.6	37.5%
32	Linear full rank	$l = 4, m = 4$	$10^{-4}$	103	59	1.75	42.72%
33	Linear rank-1	$l = 3, m = 4$	$10^{-2}$	43,507	5,382	8.08	87.63%
34	Linear rank-1 with zero column and rows	$l = 2, m = 4$	$10^{-4}$	43,690	54,613	0.8	-25.00%
35	Chebyquad	$l = 2, m = 2$	$10^{-4}$	>37,374	5,283	*	*
36	SHCB: Six hump camel	$l = 2, m = 1$	$10^{-4}$	132	88	1.5	33.33%
37	THCB: Three hump camel	$l = 2, m = 1$	$10^{-4}$	54	42	1.29	22.22%
38	BR: Branin	$l = 2, m = 1$	$10^{-4}$	7	7	1.00	0.00%
39	L3: Levy	$l = 2, m = 1$	$10^{-4}$	158	259	0.61	-63.92%
40	L5: Levy	$l = 2, m = 1$	$10^{-4}$	36	169	0.21	-369.44%
41	L8: Levy	$l = 3, m = 1$	$10^{-4}$	8	26	0.31	-225%
42	L18: Levy	$l = 7, m = 1$	$10^{-4}$	7	18	0.39	-157.14%
43	Schw2.5: Booth	$l = 2, m = 1$	$10^{-4}$	6	7	0.86	-16.67%
44	Schw2.18: Matyas	$l = 2, m = 1$	$10^{-4}$	>	28	*	*
45	Schw3.1: Schwefel	$l = 3, m = 1$	$10^{-4}$	3	7	0.43	-133.33%
46	Schw3.1p: Schwefel	$l = 3, m = 1$	$10^{-4}$	40,837	40,837	1	0.00%
47	Schw3.2: Schwefel	$l = 3, m = 1$	$10^{-4}$	12	12	1.00	0.00%
48	Griew5: Griewank	$l = 5, m = 1$	$10^{-4}$	>51,910	32	*	*
49	GP: Goldstein and price	$l = 2, m = 1$	$10^{-4}$	5,580	8,101	0.69	-45.18%
50	R4: Ratz	$l = 2, m = 1$	$10^{-4}$	92	76	1.21	17.39%

TABLE 4.15. Domains and computed global minimums and minimizers with termination criterion A.

S.No	Test function	Domain	Global minimum	Global minimizer
1	Rosenbrock	$[-1.5, 1.5]^2$	0.0000	$[0.9997, 1.0002]^2$
2	Freudenstein and Roth	$[-7.5, 7.5]^2$	0.0000	$[4.9987, 4.9989]$ $[3.9999, 4.0001]$
3	Powell badly scaled	$[-1, 1], [0, 10]$	0.0000	$[0.0000, 0.0001]$ $[9.1052, 9.1053]$
4	Brown badly scaled	$[10^{-6}, 10^6], [-1, 1]$	0.0000	$[9.9999, 10] \times 10^5$ $[0.0000, 0.0001] \times 10^{-5}$
5	Beale	$[-4.5, 4.5]^2$	0.0000	$[3.0058, 3.0081]$ , $[0.5009, 0.5032]$
6	Jenrich and Sampson	$[0.4, -0.4]^2$	124.3621...	$[0.2578, 0.2579]^2$
7	Helical valley	$[0.001, 1.5], [-1.5, -1.5]^2$	0.0000	$[0.9993, 1.0001]$ , $[0.0000, 0.0004]$ $[0.0000, 0.0008]$
8	Bard	$[-2.5, 2.5], [0.01, 2.5]^2$	0.0081...	$[0.0820, 0.0823]$ , $[1.1291, 1.1295]$ $[2.3469, 2.3472]$
9	Gaussian	$[-1.5, 1.5]^3$	0.0000	$[0.3984, 0.4014]$ , $[0.9960, 1.0020]$ $[0.000, 0.0059]$
10	Meyer	$[5.6, 5.62] \times 10^{-4}$ , $[6181.3, 6181.35]$ $[250, 350]$	87.9414...	$[0.0000, 0.0001] \times 10^3$ , $[6.1813, 6.1814] \times 10^3$ $[0.3452, 0.3453] \times 10^3$
11	Gulf research	$[0.1, 5], [0, 5]^2$	0.0000....	$[1.9375, 2.0141]$ , $[2.5000, 2.5391]$ , $[0.5620, 1.1954]$
12	Box three dimensional	$[-20, 20], [1, 20]^2$	0.0000	$[0.8984, 0.9034]$ , $[10.3515, 10.3563]$ $[1.0371, 1.0418]$
13	Powell singular	$[-3, 3]^4$	0.0000	$[0.0000, 0.0008]$ $[0.0000, 0.0015]^3$
14	Wood	$[-3, 3]^4$	0.0000	$[0.9997, 1.0002]^4$
15	Kowalik and Osborne	$[-0.1, 0.2]^4$	$0.2722... \times 10^{-3}$	$[0.1921, 0.9300]$ , $[0.1867, 0.1876]$ $[0.1195, 0.1204]$ , $[0.1335, 0.1344]$
16	Brown and Dennis	$[-25, 25]^4$	$8.5822... \times 10^4$	$[-11.5945, -11.5944]$ , $[13.2036, 13.2037]$ , $[-0.4035, -0.4034]$ , $[0.2367, 0.2368]$



Table 4.15 (Contd.)

S.No	Test function	Domain	Global minimum	Global minimizer
17	Osborne 1	$[0.5, 1.5], [1, 1.5],$ $[-1, 1.5],$ $[0.05, 1.5]^2$	0.6188...	$[1.0000, 1.0313], [10.000, 10.0313]$ $[1.0000, 1.0313], [5.0000, 5.0313]$ $[0.10433, 0.1044]$
18	Biggs EXP 6	$[1, 1.5],$ $[10, 10.5],$ $[1, 1.5][5, 5.5],$ $[4, 4.5], [3, 3.5]$	0.0000	$[1.0000, 1.0313], [10.000, 10.0313]$ $[1.0000, 1.0313], [5.0000, 5.0313]$ $[4.0000, 4.0313],$ $[3.0000, 3.0313]$
19	Osborne 2	Given at the end of the table	1.4059...	Given at the end of the table
20	Watson	$[0, 0.4]^6$	2.3864...	$[0.0000, 0.0000],$
21	Extended Rosenbrock	$[-1.2, 1.2]^4$	0.0000	$[0.4000, 0.4001]^2,$ $[0.9999, 1.0002]^4$
22	Extended Powell singular	$[-3, 3]^4$	0.0000	$[0.0000, 0.0008]^2,$ $[0.0000, 0.0015]^2$
23	Penalty I	$[-4, 4]^4$	$0.0222... \times 10^{-3}$	$[0.2499, 0.2540]^3, [0.2499, 0.2579]$
24	Penalty II	$[-0.5, 0.5]^4$	$0.0093... \times 10^{-3}$	$[0.1992, 0.2002], [0.1953, 0.1973]$ $[0.4902, 0.4922], [0.4960, 0.4981]$
25	Variably dim.	$[-1.5, 1.5]^2$	0.0000	$[0.9990, 1.0020]^2$
26	Trigonometric	$[-0.25, 0.25]^4$	0.0000	$[0.0000, 0.0040]^4$
27	Brown almost linear	$[-2.5, 2.5]^4$	0.0000	$[1.0009, 1.0022], [0.9960, 0.9974]^2$ $[1.0058, 1.0071]$
28	Discrete bound. value	$[-0.5, 0.5]^2$	0.0000	$[-0.1290, -0.1269], [-0.1602, -0.1562]$
29	Discrete integral eq.	$[-1, 1]^4$	0.0000	$[-0.1016, -0.0976], [-0.1485, -0.1445]$ $[-0.1749, -0.1679], [-0.1329, -0.1249]$
30	Broyden tridiagonal	$[-1, 1]^4$	0.0000	$[-0.5547, -0.5541], [-0.6397, -0.6391]$ $[-0.5918, -0.5908], [-0.4161, -0.4150]$
31	Broyden branded	$[-1, 1]^4$	0.0000	$[-0.4288, -0.4282], [-0.4766, -0.4755]$ $[-0.5206, -0.5195]^2$
32	Linear full rank	$[-1, 1]^4$	0.0000	$[-1.0000, -0.9980]^4$

Table 4.15 (Contd.)

33	Linear rank-1	$[0.01,1]^3$	0.4236...	$[0.2497, 0.2503]$ , $[0.0187, 0.0192]$ $[0.0467, 0.0473]$
34	Linear rank-1 with zero column and rows	$[-1,1]^2$	2.1999...	$[0.0000, 0.0001]$ , $[0.1999, 0.2001]$
35	Chebyquad	$[-2,2]^2$	0.0000	$[-0.5782, -0.5742]$ , $[0.5781, 0.5821]$
36	SHCB: Six hump camel	$[-2,2]^2$	-1.0137...	$[0.0898, 0.0898]$ , $[-0.7127, -0.7126]$
37	THCB: Three hump camel	$[-3,3]^2$	$0.1288... \times 10^{-4}$	$[0.0000, 0.0015]$ , $[0.0000, 0.0015]$
38	BR: Branin	$[-5,10],[0,15]$	0.3978...	$[-3.1421, -3.1396]$ , $[12.2753, 12.2779]$
39	L3: Levy	$[-10,10]^2$	-176.5419...	$[4.9764, 4.9765]$ , $[4.8580, 4.8581]$
40	L5: Levy	$[-10,10]^2$	-176.1377...	$[-1.3069, -1.3068]$ , $[-1.4249, -1.4248]$
41	L8: Levy	$[-10,10]^3$	0.0000	$[0.9960, 1.0059]$ , $[0.9960, 1.0157]^2$
42	L18: Levy	$[-10,10]^7$	0.0000	$[0.9997, 1.0010]$ , $[0.9985, 1.0010]^6$
43	Schw2.5: Booth	$[-5,5]^2$	0.0000	$[2.3315, 2.3340]$ , $[0.3320, 0.3370]$
44	Schw2.18: Matyas	$[-30,30]^2$	$-0.2093... \times 10^{-4}$	$[0.0219, 0.0239]^2$
45	Schw3.1: Schwefel	$[-10,10]^3$	0.0000	$[0.9985, 1.0010]^3$
46	Schw3.1p: Schwefel	$[-10,10]^3$	0.0000	$[1.0009, 1.0016]^3$
47	Schw3.2: Schwefel	$[-1.89, 1.89]^3$	0.0000	$[0.9966, 0.9986]$ , $[0.9966, 1.0004]^2$
48	Griew5: Griewank	$[-600,600]^5$	0.0000	$[0.0000, 0.0092]^5$
49	GP: Goldstein and price	$[-2,2]^2$	2.9999...	$[0.0000, 0.0001]$ , $[-1.0001, -1.0000]$
50	R4: Ratz	$[-3,3]^2$	-0.1070...	$[-0.0002, 0.0000]$ , $[-1.4576, -1.4574]$

Note: the initial domain for test function number 19, Osborne 2, is  $[1.2, 1.4]$ ,  $[0.6, 0.75]$ ,  $[0.6, 0.75]$ ,  $[0.7, 0.75]$ ,  $[0.5, 0.61]$ ,  $[2.9, 3.1]$ ,  $[4.9, 5.1]$ ,  $[6.9, 7.1]$ ,  $[1.9, 2.1]$ ,  $[4.4, 4.6]$ ,  $[5.4, 5.6]$  and the minimizer is  $[1.1999, 1.1000]$ ,  $[0.5999, 0.6000]$ ,  $[0.7171, 0.7178]$ ,  $[0.7499, 0.7501]$ ,  $[0.5412, 0.5417]$ ,  $[3.1000, 3.1001]$ ,  $[4.9000, 4.9000]$ ,  $[6.9000, 6.9000]$ ,  $[2.1000, 2.1000]$ ,  $[4.5265, 4.5270]$ ,  $[5.5718, 5.5727]$

# 5

## An improved algorithm for set inversion

### 5.1 Introduction

Let  $f = (f_1, \dots, f_m)$  be a nonlinear function from  $\mathbb{R}^l$  to  $\mathbb{R}^m$ . Let  $\mathcal{Y}$  be a subset of  $\mathbb{R}^m$ . Then, the problem of set inversion can be posed as the characterization of

$$\mathcal{S} = \left\{ x \in \mathbb{R}^l : f(x) \in \mathcal{Y} \right\} = f^{-1}(\mathcal{Y}) \quad (5.1)$$

In this work, we address the problem of characterizing  $\mathcal{S}$  defined by a set of nonlinear inequalities

$$\mathcal{S} = \left\{ x \in \mathbb{R}^l : f(x) > 0 \right\} = f^{-1}([0, \infty[^m) \quad (5.2)$$

Such problems arise, for instance, in robust stability analysis of feedback control systems [1]. Jaulin *et al.* [31] proposed an algorithm for solving the above problem via interval analysis. The algorithm has also been successfully tested in applications, such as characterization of robust stability domains, parameter and state estimation, and robotics.

In this work, we propose some improvements to the algorithm of Jaulin *et al.* for characterization of  $\mathcal{S}$  via set inversion. We shall assume that  $f$  is continuously differentiable on  $\mathbf{X}$ . The proposed improvements are based on exploiting the property of monotonicity in two ways:

- The powerful monotonicity test form is used as an inclusion function for  $f$ , and
- If  $f_j$ ,  $j = 1, \dots, m$  is found to be monotonically *increasing* resp. decreasing in *every* component direction on a given box, then the part of box where the inequality  $f_j > 0$  is certainly infeasible is found and discarded.

Further, only one box is processed in each iteration of the algorithm of Jaulin *et al.* Such processing is inherently slow, due to its sequential nature. On the other hand, in the proposed

algorithm, *all* boxes present in the list at each iteration are processed concurrently using *vectorization*. Clearly, *vectorization* does not alter in any way the essence of the algorithm. It however helps to reduce the processing overheads and leads to better memory management in the processor.

We test then test and compare the performances of the proposed and existing algorithms on two robust stability problems, including a case study of speed control of a jet engine.

## 5.2 Basic algorithm

Let  $\mathbf{X}^0 \in I(\mathbb{R}^l)$  be the initial feasible box. The algorithm of Jaulin *et al.* [31], [79] encloses the portion of  $\mathcal{S}$  contained in  $\mathbf{X}^0$ , between two partitions  $K_{in}$  and  $K_{out}$  in the sense that

$$K_{in} \subseteq \mathbf{X}^0 \cap \mathcal{S} \subseteq K_{out} \quad (5.3)$$

where,  $K_{out} = K_\varepsilon \cap K_{in}$  and  $K_\varepsilon$  is a list of all indeterminate boxes.

The algorithm of Jaulin *et al.* is as follows.

Inputs: The initial box  $\mathbf{X}^0$ , natural inclusion function  $F$ , and an accuracy parameter  $\varepsilon_x$ .

Outputs: A list  $K_{in}$  of all boxes guaranteed to belong to  $\mathcal{S}$ , and a list  $K_{out} = K_\varepsilon \cup K_{in}$ .

BEGIN Algorithm

1. Initialize  $\mathbf{X} = \mathbf{X}^0, K_{in} = \{\}, K_{out} = \{\}, L = \{\mathbf{X}\}$ .
2. Remove the first box  $\mathbf{X}$  from list  $L$  and evaluate  $F(\mathbf{X})$ .
3. If  $\inf F(\mathbf{X}) > 0$  for all  $i = 1, \dots, m$  then deposit  $\mathbf{X}$  in lists  $K_{in}$  and  $K_{out}$ , and go to step 7.
4. If  $\sup F_i(\mathbf{X}) \leq 0$  for any  $i = 1, \dots, m$  then go to step 7.
5. If  $w(\mathbf{X}) < \varepsilon_x$  then deposit  $\mathbf{X}$  in list  $K_{out}$  and go to step 7.
6. Bisect  $\mathbf{X}$  in maximum width coordinate direction  $k$ , getting boxes  $\mathbf{V}^1, \mathbf{V}^2$  such that  $\mathbf{X} = \mathbf{V}^1 \cup \mathbf{V}^2$ . Deposit these subboxes in  $L$ .
7. If the list  $L$  is empty, EXIT algorithm. Else go to step 2.

END Algorithm

## 5.3 Proposed Algorithm

### 5.3.1 Throwing parts of box (TPB)

Let  $\mathbf{X} \subseteq \mathbf{X}^0$ . Let  $f$  be a single-valued function, in this subsection. Suppose that  $f$  is found to be monotonically *increasing* in *every* component direction on  $\mathbf{X}$ . Then, Algorithm TPB

given below locates the subbox  $\mathbf{C} \subseteq \mathbf{X}$  on which the inequality  $f > 0$  is certainly infeasible, and outputs a list  $L_X$  of boxes whose union is the complement of  $\mathbf{C}$  in  $\mathbf{X}$ .

**Algorithm TPB** ( $f, \mathbf{X}, L_X$ )

Inputs: The function  $f$  and box  $\mathbf{X}$ . It is assumed that  $f$  is monotonically *increasing* in every component direction on  $\mathbf{X}$ .

Outputs: A list  $L_X$  such that  $\bigcup_{\mathbf{W} \in L_X} \mathbf{W} = \mathbf{X} \setminus \mathbf{C}$ .

1. Check for the trivial cases:
  - (a) If  $f(\underline{\mathbf{X}}_1, \dots, \underline{\mathbf{X}}_l) > 0$  then the inequality  $f > 0$  is certainly feasible on entire  $\mathbf{X}$ .  
Set  $L_X \leftarrow \{\mathbf{X}\}$  and RETURN.
  - (b) If  $f(\overline{\mathbf{X}}_1, \dots, \overline{\mathbf{X}}_l) \leq 0$  then the inequality  $f > 0$  is certainly infeasible on entire  $\mathbf{X}$ .  
Set  $L_X \leftarrow \{\}$  and RETURN.
2. Parametrize the line joining points  $(\underline{\mathbf{X}}_1, \dots, \underline{\mathbf{X}}_l)$  and  $(\overline{\mathbf{X}}_1, \dots, \overline{\mathbf{X}}_l)$  in terms of a single parameter  $\lambda$ :

$$(\lambda \overline{\mathbf{X}}_1 + (1 - \lambda) \underline{\mathbf{X}}_1, \dots, \lambda \overline{\mathbf{X}}_l + (1 - \lambda) \underline{\mathbf{X}}_l), \lambda \in [0, 1]$$

3. Using a nonlinear solver such as Newton - Raphson, find<sup>1</sup>  $\lambda^* \in [0, 1]$  such that  $f(\lambda^*) = 0$ .
4. Construct a subbox  $\mathbf{C} \subseteq \mathbf{X}$  on which the inequality  $f > 0$  is certainly infeasible:

$$\mathbf{C} = ([\underline{\mathbf{X}}_1, \lambda^* \overline{\mathbf{X}}_1 + (1 - \lambda^*) \underline{\mathbf{X}}_1], \dots, [\underline{\mathbf{X}}_l, \lambda^* \overline{\mathbf{X}}_l + (1 - \lambda^*) \underline{\mathbf{X}}_l])$$

5. Using the box complementation algorithm in [34], find the complement of box  $\mathbf{C}$  in  $\mathbf{X}$  to get a list  $L_X$  such that

$$\bigcup_{\mathbf{W} \in L_X} \mathbf{W} = \mathbf{X} \setminus \mathbf{C}$$

END Algorithm.

On similar lines, the above algorithm can be given for the case where  $f$  is monotonically *decreasing* in all component directions on  $\mathbf{X}$ . The boxes in list  $L_X$  are sent for further processing to the main algorithm.

---

<sup>1</sup>Since  $f$  is monotonically increasing on  $\mathbf{X}$  by hypothesis, and since the trivial cases have been dealt with at an earlier step,  $\lambda^*$  necessarily exists and is unique on the line.

### 5.3.2 The algorithm

The proposed algorithm uses the monotonicity test form  $F_{MT}$  instead of the natural inclusion function  $F$ . Further, as explained earlier, *all* boxes that are present in the list are processed concurrently at each iteration using *vectorization*. This can be done without altering in any way the essence of the set inversion algorithm.

We next present the proposed algorithm.

#### Proposed algorithm

Inputs: The initial box  $\mathbf{X}^0$ , monotonicity test form  $F_{MT}$ , and an accuracy parameter  $\varepsilon_x$ .

Outputs: A list  $K_{in}$  of all boxes guaranteed to belong to  $\mathcal{S}$ , and a list  $K_{out} = K_\varepsilon \cup K_{in}$  enclosing  $\mathcal{S}$ .

BEGIN Algorithm

1. Initialize  $\mathbf{X} = \mathbf{X}^0, K_{in} = \{\}, K_{out} = \{\}, L = \{\mathbf{X}\}$ .
2. Remove all boxes from list  $L$  and evaluate  $F_{MT}$  over all the boxes.
3. Deposit all boxes for which  $\inf F(\mathbf{X}) > 0$  for all  $i = 1, \dots, m$  in the lists  $K_{in}$  and  $K_{out}$ .
4. Discard all those remaining boxes for which  $\sup F_i(\mathbf{X}) \leq 0$  for any  $i = 1, \dots, m$ .
5. Deposit all those remaining boxes for which  $w(\mathbf{X}) < \varepsilon_x$  in list  $K_{out}$ .
6. Find all those boxes for which  $f_i$  is monotonically increasing resp. decreasing in every direction,  $i = 1, \dots, m$ . Apply Algorithm TPB to discard infeasible parts of these boxes.
7. Bisect all boxes in the maximum width coordinate direction  $k$ , getting subboxes  $\mathbf{V}^1, \mathbf{V}^2$  such that  $\mathbf{X} = \mathbf{V}^1 \cup \mathbf{V}^2$ . Deposit all these subboxes in  $L$ .
8. If the list  $L$  is empty, EXIT algorithm. Else, go to step 2.

END Algorithm

**Remark 5.1** *In the above algorithm, function evaluations, zero exclusion checks, width checks, and bisections on all boxes in an iteration are performed using vectorized interval arithmetic operations.*

## 5.4 Test example: Polynomial stability

Consider the polynomial [79]

$$p(s, x) = s^3 + \sin(x_1 x_2) s^2 + x_1^2 s + x_1 x_2 \quad (5.4)$$

We wish to find the set of  $(x_1, x_2)$  values for which the polynomial in (5.4) is Hurwitz stable [8]. Necessary and sufficient conditions for the Hurwitz stability of the polynomial can be

obtained, for instance, by enforcing positivity in the first column of the Routh table [8]. This give the set of inequalities

$$\begin{aligned} f_1(x_1, x_2) &= x_1 x_2 > 0 \\ f_2(x_1, x_2) &= \sin(x_1 x_2) > 0 \\ f_3(x_1, x_2) &= x_1^2 \sin(x_1 x_2) - x_1 x_2 > 0 \end{aligned} \quad (5.5)$$

We solve the inequalities using the existing and proposed set inversion algorithms of interval analysis, with the initial box as  $\mathbf{X}^0 = ([-10, 10], [-10, 10])$ . We carry out the computations on a Sun 440 MHz Ultra Sparc 10 machine, with 1 GB RAM using Forte FORTRAN 95 [73]. To compare the performances of the proposed and existing set inversion algorithms, we choose the accuracy  $\varepsilon_x = 10^{-3}$ , and use the following as performance metrics:

- Computational time taken by the algorithm.
- Number of boxes in list  $K_\varepsilon$  for which system stability is indeterminate.
- Maximum length of list  $L$  in the algorithm.

Table 5.1 reports the values of the various performance metrics. Although the proposed algorithm is found to be somewhat slower than the existing algorithm, it requires less space complexity in terms of a smaller list length and produces a lesser number of indeterminate boxes.

Fig. 5.1 shows the zoomed plot of the domain of guaranteed stability generated with the algorithms. From this plot, we see that the proposed algorithm gives a larger region for which the system stability is guaranteed. Similarly, Fig. 5.2 shows the zoomed plot of the domain of indeterminate stability generated with the algorithms. From these plots, we find that the proposed algorithm gives a smaller region for which the system stability is indeterminate. Thus, the proposed algorithm encloses the domain  $\mathcal{S}$  of robust stability more accurately.

## 5.5 Case study: Speed control of jet engine

### 5.5.1 Vertex-type procedure

All real world systems are subject to various disturbances and uncertainties. These uncertainties can be represented as variations in coefficients of transfer functions in the Laplace domain, to form an *interval* plant family.

Consider a strictly proper *interval* plant family  $\mathcal{P}$  of the form

$$p(s, q, r) = \frac{n_p(s, q)}{d_p(s, r)} = \frac{q_0 + q_1 s + \dots + q_l s^l}{r_0 + r_1 s + \dots + s^k}; k > l \quad (5.6)$$



where interval bounds are *a priori* given for each uncertain coefficient  $q_i$  and  $r_i$ . Let

$$c(s, a, b) = \frac{n_c(s, a)}{d_c(s, b)} = \frac{a_0 + a_1s + \dots + a_ms^m}{b_0 + b_1s + \dots + s^n}; n > m \quad (5.7)$$

be a compensator in a feedback structure for the interval plant. If  $c(s, a, b)$  is such that it stabilizes the entire  $\mathcal{P}$ , then  $c(s, a, b)$  is said to *robustly* stabilize  $\mathcal{P}$ .

For interval plants of form (5.6), Nataraj and Srivastava [59] proposed a ‘vertex - type’ approach to find the set of all parameter values of a robustly stabilizing compensator with specified structure. By a vertex-type approach, we mean the following. From the interval plant  $\mathcal{P}$ , a small set of special polynomials is derived. This small set has the distinguishing property that if a compensator  $c(s, a, b)$  is found such that all constituent polynomials are stabilized, then the same  $c(s, a, b)$  would robustly stabilize the original interval plant  $\mathcal{P}$ .

The basis of the vertex-type approach is provided by the Generalized Kharitonov theorem of Bhattacharyya et al. [8], which states that the robust stability of the closed loop system having an interval plant  $\mathcal{P}$  is equivalent to that of the set of 32 associated so-called Generalized Kharitonov Segments (GKS). However, in contrast with the approach of Bhattacharyya *et al.* where the entire GKS is considered, here we adopt a vertex-type approach, which in general is more computationally tractable. What we do is to take up each GKS in turn, and construct virtual vertices whose stabilization by compensator  $c(s, a, b)$  is sufficient to ensure the same for the GKS. We next set up Routh tables for all the obtained vertex polynomials, and then derive inequality constraints in terms of the compensator parameters, so as to enforce positivity in the first column of each Routh table. Finally, we solve the total set of inequalities using the set inversion via interval analysis technique, to get the set of all compensator parameters that robustly stabilize the set of GKS. By the Generalized Kharitonov theorem, the same set of compensator parameters would also robustly stabilize the interval plant  $\mathcal{P}$ .

### 5.5.2 Obtaining compensator parameters

The control logic of the modern day jet engine is comprised of many control loops, each of which has a specific purpose. Typical control loops include compressor speed governor, an acceleration and a de-acceleration loop, and various limiting loops for temperature, speed, fuel flow, and rate of change of fuel flow. A block diagram of a typical compressor speed control loop is shown in Fig. 5.3. A compressor speed demand schedule establishes the desired compressor speed as a function of inlet temperature and throttle position. The compressor speed error is determined from the difference in the desired and actual speeds. The desired performance requirement is to obtain a fast response without any overshoot.

In the present work, our aim is to determine if a robustly stabilizing compensator of a desired structure exists for an interval plant model of a jet engine. If it exists, then we wish to characterize the set of stabilizing compensator parameters for the given plant model. Set inversion via interval analysis is used for this purpose.

Consider the single- input and single-output transfer function model of a jet engine under development in India, with the manipulated variable as main burner fuel flow and the controlled variable as compressor speed:

$$p(s, \alpha) = \frac{n_p(s)}{d_p(s, \alpha)} = \frac{4.3}{\alpha_3 s^3 + \alpha_2 s^2 + \alpha_1 s}; \alpha_3 \in [0.004, 0.005], \alpha_2 \in [0.4, 0.5], \alpha_1 = 1 \quad (5.8)$$

Note that the above defines an *interval* plant model. Now, we wish to synthesize a lead - lag compensator of the form

$$c(s, a, b, c) = \frac{n_c(s)}{d_c(s)} = \frac{a(1 + bs)}{1 + cs} \quad (5.9)$$

We seek the set of all parameter values of the above compensator that robustly stabilize the interval plant in (5.8).

We follow the vertex-type control design procedure outlined in previous subsection. There are four GKS in this case:

$$\begin{aligned} & 4.3a(1 + bs) + (1 + cs) [(0.001\lambda + 0.004) s^3 + 0.5s^2 + s] \\ & 4.3a(1 + bs) + (1 + cs) [(0.001\lambda + 0.004) s^3 + (0.1\lambda + 0.4) s^2 + s] \\ & 4.3a(1 + bs) + (1 + cs) [(-0.001\lambda + 0.005) s^3 + (0.1\lambda + 0.4) s^2 + s] \\ & 4.3a(1 + bs) + (1 + cs) [(-0.001\lambda + 0.005) s^3 + 0.4s^2 + s] \end{aligned}$$

For each of these GKS, we find the distinguishing vertex polynomials of the vertex-type approach. For example, the 4 vertex polynomials corresponding to the first GKS are

$$\begin{aligned} & 0.004s^4 + (0.5c + 0.004) s^3 + (0.5 + c) s^2 + (1 + 4.3ab) s + 4.3a \\ & 0.005s^4 + (0.5c + 0.005) s^3 + (0.5 + c) s^2 + (1 + 4.3ab) s + 4.3a \\ & 0.004s^4 + (0.5c + 0.005) s^3 + (0.5 + c) s^2 + (1 + 4.3ab) s + 4.3a \\ & 0.005s^4 + (0.5c + 0.004) s^3 + (0.5 + c) s^2 + (1 + 4.3ab) s + 4.3a \end{aligned}$$

Similarly, 4 vertex polynomials for each of the other 3 GKS are also determined. This gives a total of 16 vertex polynomials. Next, Routh tables are constructed for all these 16 vertex polynomials. Enforcement of the positivity requirement for the first column of each Routh table leads to a total set of 80 nonlinear inequality constraints in the compensator parameters. An additional constraint of  $a = 0.7$  is enforced for the steady state performance requirement of the control system.

We solve the inequalities for the compensator parameter values, using the proposed and existing set inversion algorithms of interval analysis. The initial box is taken as  $a = 0.7$ ,  $b \in [0.01, 1]$ ,  $c \in [0.01, 8]$ . We carry out the computations on a Sun 440 MHz Ultra Sparc 10 machine with 1 GB RAM, using Forte FORTRAN 95 compiler [73]. To compare the

performances of the existing and proposed set inversion algorithms, we choose two values for the accuracy  $\varepsilon_x = 10^{-2}$  and  $10^{-3}$ , and use the following as performance metrics:

- Computational time taken by the algorithm.
- Number of boxes in list  $K_\varepsilon$  for which system stability is indeterminate.
- Maximum length of list  $L$  required in the algorithm.

Table 5.2 reports the values of the various performance metrics. Although the proposed algorithm is found to be somewhat slower than the existing algorithm, it requires less space complexity in terms of a smaller list length and produces a lesser number of indeterminate boxes.

The last point made above is confirmed from Fig. 5.4 which shows a zoomed plot of the domain of guaranteed stability generated with the algorithms. From this plot, we see that the proposed algorithm gives a larger domain of compensator parameter values for which the system stability is guaranteed. Similarly, Fig. 5.5 shows a zoomed plot of the domain of indeterminate stability generated with the algorithms. From this plot, we find that the proposed algorithm gives a smaller region of compensator parameter values for which the system stability is indeterminate. Thus, the proposed algorithm encloses the domain  $\mathcal{S}$  of robust stability more accurately.

As an interesting further control study, we arbitrarily select one set of compensator parameter values from the feasible region in Fig. 5.4, as  $a = 0.7$ ,  $b = 0.5074$ ,  $c = 0.1062$ . For these values of the compensator parameters, all roots of the 16 vertex polynomials described earlier are found to lie in the left half of the complex  $s$ -plane, thereby confirming stability. Fig. 5.6 shows the closed loop system responses to a unit step in the compressor speed command input with this compensator. The responses are found for various plant models from the interval plant family. We observe that no overshoot occurs in the compressor speed output - this is an important performance requirement in the jet engine speed control loop. We also find the settling time of the responses to be quite good, varying between 0.9 and 1.2 seconds over the interval plant. Overall, the designed control system is found to perform quite satisfactorily.

## 5.6 Conclusions

We presented an improved interval analysis algorithm for characterizing the domain of a set of nonlinear inequalities. The improvements were based on the powerful tool of monotonicity. For two robust stability problems, including a case study of control of a jet engine, we found that the proposed algorithm encloses the stability domain more accurately, requires smaller list lengths, but takes more computational time.

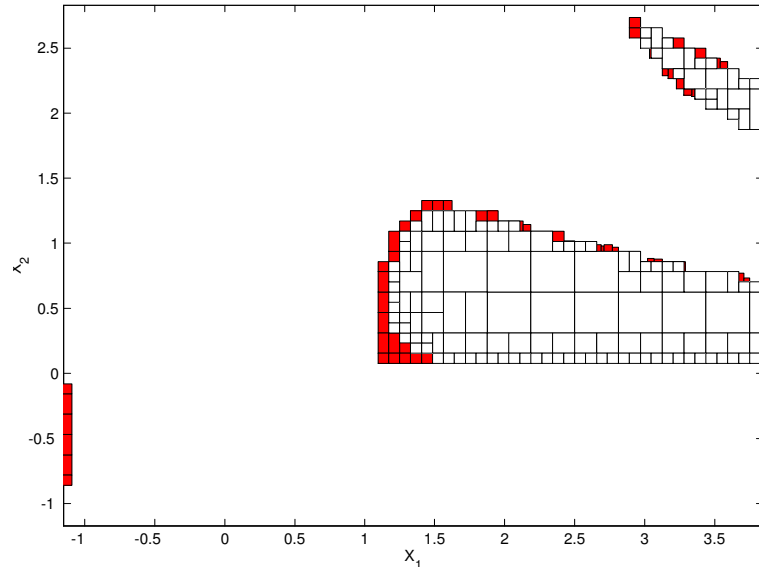


FIGURE 5.1. A zoomed plot of the domain of guaranteed robust stability for the polynomial example. The values are obtained from the list  $K_{in}$ . The shaded part is the extra domain given by the proposed set inversion algorithm.

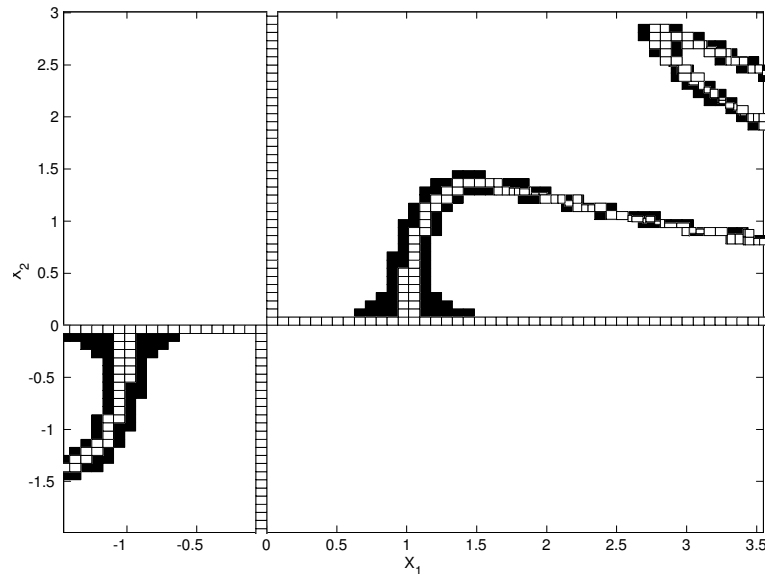


FIGURE 5.2. A zoomed plot of the domain of indeterminate robust stability for the polynomial example. The values are obtained from the boxes in  $K_\varepsilon$ . The shaded part is the extra region given by the existing set inversion algorithm of Jaulin *et al.*

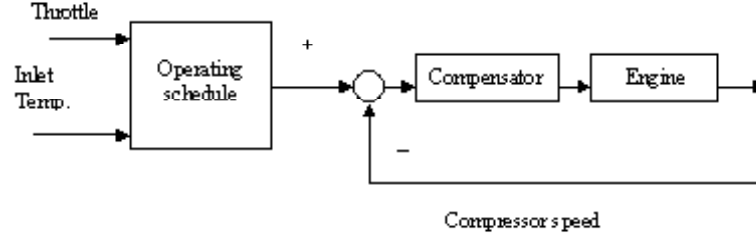


FIGURE 5.3. Block diagram of compressor speed control loop of jet engine.

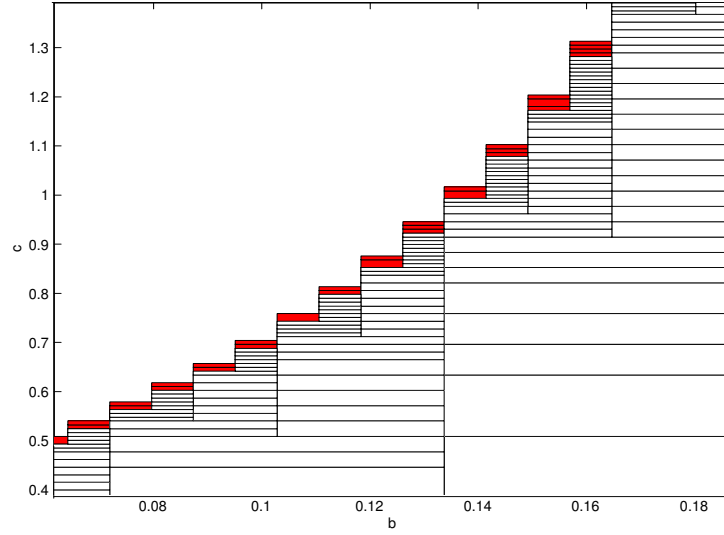


FIGURE 5.4. Zoomed plot of the domain of guaranteed robust stability for the speed control loop of jet engine. The values are obtained from the list  $K_{in}$ . The shaded part is the extra region given by the proposed set inversion algorithm. Here,  $\varepsilon_x = 10^{-3}$  and  $a = 0.7$ .

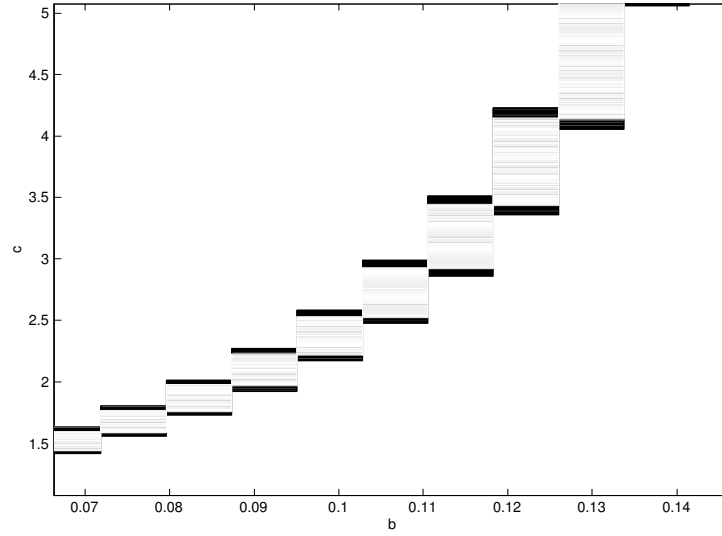


FIGURE 5.5. Zoomed plot of the domain of indeterminate robust stability for the speed control loop of jet engine. The values are obtained from boxes in  $K_\varepsilon$ . The shaded part is the extra region given by existing set inversion algorithm of Jaulin *et al.* Here,  $\varepsilon_x = 10^{-3}$  and  $a = 0.7$ .

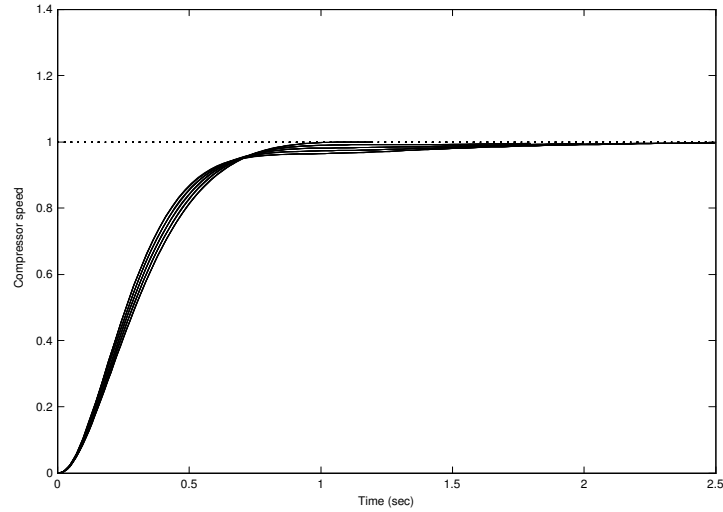


FIGURE 5.6. Responses of the compressor speed control system to a unit step in the compressor speed command input. The compensator parameter values are chosen from the solution set obtained with the proposed set inversion algorithm. The responses are given for various plants in the interval plant family.

TABLE 5.1. Performance comparison of set inversion algorithms for characterization of the domain of robust stability for a test polynomial.

Performance metric	Algorithm	
	Existing [31]	Proposed
time (s)	80.5	99.5
$K_\varepsilon$	644,602	426,871
Max. list length	1,043,560	151,590

TABLE 5.2. Performance comparison of set inversion algorithms for the jet engine control problem.

Accuracy	Performance metric	Algorithm	
		Existing [31]	Proposed
$\varepsilon_x = 0.01$	time (s)	1.6	9.8
	$K_\varepsilon$	1156	939
	Max. list length	2064	1854
$\varepsilon_x = 0.001$	time (s)	13.1	77.4
	$K_\varepsilon$	9279	7504
	Max. list length	16,674	14,861

# 6

## A new rule for bisection direction selection

### 6.1 Introduction

A fundamental problem in numerical analysis is that of computing the range  $\bar{f}(\mathbf{X})$ . Since in general it is not possible to compute the exact range  $\bar{f}(\mathbf{X})$ , we therefore consider the problem of finding an interval enclosure of  $\bar{f}(\mathbf{X})$  with a desired degree of accuracy  $\varepsilon$ . Interval analysis [4], [49] provides several techniques to solve this range computation problem. A recently proposed interval analysis based algorithm for range computations, shown to be usually more efficient than other similar algorithms, is as follows.

**A Model algorithm for computing range enclosures** [58].

*Inputs:* An inclusion function  $F$  of  $f$ , the initial box  $\mathbf{X}^0 \in I(\mathbb{R}^l)$ , and the specified accuracy  $\varepsilon$ .

*Output:* An enclosure of the exact range  $\bar{f}(\mathbf{X}^0)$  having the specified accuracy  $\varepsilon$ .

BEGIN Algorithm

1. Set  $\mathbf{X} = \mathbf{X}^0$ , initialize the working list  $L = \{\mathbf{X}\}$  and solution list  $L_{sol} = \{\}$ .
2. Remove all boxes from list  $L$ .
3. Choose a component direction  $k$  for bisection, based on a rule for bisection direction selection (see below).
4. Bisect all boxes in component direction  $k$ .
5. Evaluate  $F$  over all the subboxes obtained from above bisection.
6. Deposit all evaluations of  $F$  whose widths are at most  $\varepsilon$  in the solution list  $L_{sol}$ , and discard the corresponding subboxes from further processing.



7. Deposit the remaining boxes in list  $L$ . If  $L$  is empty go to the following step, else go to step 2.
8. Take the union of all intervals present in the solution list  $L_{sol}$  to obtain the desired range enclosure.

END Algorithm.

The selection of the bisection direction in step 3 has a significant effect on the efficiency of the algorithm, both in terms of the number of subboxes generated and the computational time taken. Among the widely used rules for bisection direction selection are the ‘maximum width’ and ‘maximum smear’ rules described below. For a more detailed exposition of the various rules for bisection direction selection, see, for instance, Csendes and Ratz [15].

In this chapter, we propose a new rule for bisection direction selection in the model algorithm for range finding. In the proposed rule, at each iteration of the above algorithm, we pick a box  $\mathbf{X}^*$  *randomly* from the current list and actually find out which is the ‘best bisection direction’ for the random box  $\mathbf{X}^*$ . That is, we bisect  $\mathbf{X}^*$  along each and every component direction and find out in which bisection direction the maximum function width over both the resulting subboxes is the *least*. We then take this ‘best bisection direction’ for  $\mathbf{X}^*$  as also the ‘best bisection direction’ for all other boxes of current list, and bisect all these boxes in the said direction. The philosophy behind the proposed rule is that “the best bisection direction for the random box of current list is (likely) the best bisection direction for all other boxes of current list”.

We next briefly outline the existing bisection direction selection rules used for performance comparison in this study.

## 6.2 Bisection Direction Selection Rules

A merit function for bisection direction selection is [15]

$$k := \min \left\{ j \mid j \in 1(1)l \text{ and } d(j) = \max_{i=1}^l d(i) \right\} \quad (6.1)$$

where  $d(i)$  is determined by the given rule. The ‘min’ appears in the above RHS, so that if  $d(i)$  is achieved along several component directions, then the smallest one is taken for bisection.

### 6.2.1 Rule A (*Maximum width*)

This is a derivative free rule and is based on the interval width, with

$$d(i) := w(\mathbf{X}_i) \quad (6.2)$$

used in (6.1). With this rule, bisection is done along the component direction of maximum interval width.

### 6.2.2 Rule B (Scaled maximum width)

We also have some scaled variants of above derivative free rule, with the following used for  $d(i)$  in (6.1):

$$\begin{aligned} \text{Rule B-1: } d(i) &:= \begin{cases} w(\mathbf{X}_i) & \text{if } 0 \in \mathbf{X}_i \\ w(\mathbf{X}_i) / \langle \mathbf{X}_i \rangle & \text{otherwise} \end{cases} \\ \text{Rule B-2: } d(i) &:= \begin{cases} w(\mathbf{X}_i) & \text{if } 0 \in \mathbf{X}_i \\ w(\mathbf{X}_i) / m(\mathbf{X}_i) & \text{otherwise} \end{cases} \\ \text{Rule B-3: } d(i) &:= \begin{cases} w(\mathbf{X}_i) & \text{if } 0 \in \mathbf{X}_i \\ w(\mathbf{X}_i) / |m(\mathbf{X}_i)| & \text{otherwise} \end{cases} \end{aligned}$$

Rule B-1 is given by Csendes and Ratz [15]. With these rules, bisection is done along the component direction of maximum *scaled* interval width.

### 6.2.3 Rule C (Maximum smear)

Csendes and Ratz [15] suggest the following for  $d(i)$  in (6.1):

$$d(i) := w(F'_i(\mathbf{X})) (\mathbf{X}_i - m(\mathbf{X}_i)) \quad (6.3)$$

where,  $F'(\mathbf{X})$  is an inclusion function of the gradient  $\nabla f(x)$ . The rule corresponds to the ‘maximum smear’ rule of Kearfott in [34]. When  $\min F'_i(\mathbf{X}) = 0$  or  $\max F'_i(\mathbf{X}) = 0$ , the rule reduces to the one suggested by Walster [26].

## 6.3 A new rule for bisection direction selection

We propose a new derivative free rule for bisection direction selection. At each iteration of above model algorithm, we pick a box  $\mathbf{X}^*$  randomly from among those of current list in step 3. We then bisect  $\mathbf{X}^*$  along the *first* component direction, and find the maximum width of  $F$  over the two resulting subboxes. We next bisect  $\mathbf{X}^*$  along the *second* component direction, and find the maximum width of  $F$  over the two resulting subboxes. We then repeat this process for all the  $l$  component directions. Finally, we find the component direction of bisection corresponding to which the obtained maximum width of  $F$  is the *least*, and choose that direction as the bisection direction for all boxes of current list. We can put the above steps in the form of a procedure.

1. Randomly select a box  $\mathbf{X}^*$  from among those of current list in step 3 of model algorithm.
2. FOR  $i = 1$  to  $l$  DO

3. Bisect  $\mathbf{X}^*$  along coordinate direction  $i$  to get two subboxes  $\mathbf{V}^1$  and  $\mathbf{V}^2$  such that  $\mathbf{X}^* = \mathbf{V}^1 \cup \mathbf{V}^2$ .
4. Evaluate  $F(\mathbf{V}^1), F(\mathbf{V}^2)$ .
5. Form  $d(i) = -\max(w(F(\mathbf{V}^1)), w(F(\mathbf{V}^2)))$ .
6. END FOR
7. Use the above found  $d(i)$  in (6.1) to obtain the component direction  $k$  for bisection.

## 6.4 Application: template generation

We apply the above bisection direction selection rules in the context of robust control system design using Horowitz's Quantitative feedback theory (QFT) approach [27]. Consider a system represented by the transfer function  $g(s, \lambda)$ , where  $\lambda = \{\lambda_1, \dots, \lambda_l\}$  is a real vector of the system parameters and  $s$  is the Laplace variable. The parameters  $\lambda_i$  vary independently over given real intervals  $\Lambda_i^0$ , so that we have a box  $\Lambda^0 = \{\Lambda_1^0, \dots, \Lambda_l^0\}$  of system parameters. Denote the phase angle function  $f_1$  and magnitude function  $f_2$  of  $g(s, \lambda)$  by

$$f_1(\lambda) := \arg g(s = j\omega, \lambda); \quad f_2(\lambda) := |g(s = j\omega, \lambda)|$$

where  $j = \sqrt{-1}$ , and  $\omega$  is the frequency variable. Define the angle-magnitude function  $f$  as

$$f(\lambda) := (f_1(\lambda), f_2(\lambda))$$

The set  $\mathcal{G} := \{f(\lambda), \lambda \in \Lambda^0\}$  defines a region in the angle - magnitude plane, called as the *template* or *value set* of  $g(s, \lambda)$  at the given frequency  $\omega$ . We wish to generate a collection of angle - magnitude rectangles covering template  $\mathcal{G}$  such that each rectangle has a width at most equal to some prescribed accuracy  $\varepsilon$ . Clearly, this problem is a more involved version of the range computation problem, and is known as the template generation or value set computation problem in robust control system design, see for instance, [1], [3], [27].

We test and compare the performance of the various bisection direction selection rules in the template generation problem for ten real-world examples taken from the engineering literature. The examples are described in Appendix C. We choose the units for the magnitude as decibels (dB), and for phase angle as degrees. In these units, we specify the width of each generated rectangle as  $\varepsilon = 1$ . That is, we wish to generate the template in each example to an accuracy of 1 deg and 1 dB.

For all our computations, we use a single processor PC Pentium III 550 MHz machine with 384 MB RAM, and the interval arithmetic toolbox INTLAB of Rump [68]. For convenience, we henceforth refer to the randomly picked box  $\mathbf{X}^*$  as the *random* box.

To compare the performance of the bisection direction selection rules, we choose the following performance metrics:

- Number of solution boxes covering the template, and
- Computational time taken (seconds) to generate the template.

Table 6.1 reports the performance values for different five runs of the algorithm with the proposed bisection direction selection rule. We observe slight variations in the performance values from run to run - this is due to the ‘randomness’ involved in picking the random box. We therefore also give the average performance values over all the five runs, and use these average performance values in the following.

We next proceed to examine how well the philosophy underlying the proposed rule holds. Table 6.2 shows performance of the proposed bisection direction selection rule for a random run. The fifth column of the table gives the number of iterations  $\alpha$  where more than 90% of the boxes of the current list have the same ‘best bisection direction’ (in the sense of section 1) as that of the random box<sup>1</sup>. The last column gives the average percentage of boxes over all the iterations that have the same ‘best bisection direction’ as that of the random box. On the average, more than three-fourths of the boxes in a list are found to have the same ‘best bisection direction’ as that of the random box, a finding which, by and large, justifies the philosophy of bisection direction selection adopted in this work.

Tables<sup>2</sup> 6.3 and 6.4 give the results obtained with different bisection direction selection rules. As mentioned earlier, the results reported for the proposed bisection direction selection rule are the averages over five different runs.

At the outset, we note that with the proposed bisection direction selection rule, we are able to solve *all* the examples, whereas with bisection direction selection rules A, B-1, B-2, B-3 and C we are able to solve only 70%, 80%, 80%, 80%, and 70% of the examples, respectively.

Based on the data in the Tables, we further compare the performances of the rules using different evaluation methods: ranking, averaging, and performance profiling.

#### 6.4.1 Ranking

Tables 6.5 and 6.6 give the ranks obtained by the various rules in our tests. These tables show that the proposed bisection direction selection rule attains

- 1<sup>st</sup> rank in 80% of the examples for the number of solution boxes metric, and
- 1<sup>st</sup> rank in 90% of the examples for computational time metric.

That is, the proposed bisection direction selection rule is the most effective for both performance metrics in most examples.

---

<sup>1</sup>For each example and iteration, we actually bisected every box in every component direction to arrive at these findings.

<sup>2</sup>A ‘\*’ entry in the columns of Tables indicates that the solution could not be obtained by the algorithm for the prescribed accuracy, due to excessive requirement of memory.

### 6.4.2 Averaging

For a given metric, the percentage reduction obtained with the proposed rule compared to the considered rule (such as Rule A, B, or C) is computed as

$$\% \text{ reduction} = \frac{\text{Perf. metric with considered rule} - \text{Perf. metric with proposed rule}}{\text{Perf. metric with considered rule}} \times 100$$

In Table 6.7, we give the percentage reductions obtained with the proposed rule compared to existing rules for both metrics. We observe the following

- Number of solution boxes: With the proposed rule, the average percentage reduction over all examples compared to Rule A is 55%, Rule B-1 is 64%, Rule B-2 is 65%, Rule B-3 is 65%, and Rule C is 42%.
- Computational time: With the proposed rule, the average percentage reduction over all examples compared to Rule A is 67%, Rule B-1 is 75%, Rule B-2 is 75%, Rule B-3 is 71%, and Rule C is 79%.

That is, for both performance metrics, we obtain considerable percentage reductions with the proposed bisection direction selection rule over existing bisection direction selection rules.

### 6.4.3 Performance profiling

We plot the performance profiles for both performance metrics in Fig. 6.1. We make the following observations based on these plots:

- Number of solution boxes: Performance profile plots show that the proposed bisection direction selection rule is able to solve all the examples for  $\tau = 1$ . The existing bisection direction selection rules are able to solve only 80% of the examples for  $\tau < 63$ .
- computational time: Performance profile plots show that the proposed bisection direction selection rule takes least time and solves all the examples for  $\tau = 1$ . The existing bisection direction selection rules are able to solve 80% of the examples for  $\tau < 230$ .

## 6.5 Summary

At the outset, we note that with the proposed bisection direction selection rule we are able to solve *all* the examples, whereas with *none* of the basic rules we are able to solve all the examples. On the average, about 58% reduction in the number of solution boxes and 73% reduction in computational time is obtained with the proposed bisection direction selection rule over existing rules.

It is a noteworthy empirical finding that more than three-fourths of the boxes present at any given iteration of the algorithm had the same ‘best bisection direction’ as that of the random box. This empirical finding vindicates by and large the philosophy behind the proposed rule “the best bisection direction for a randomly picked box from a given list is likely the best bisection direction for all other boxes of the same list”. The philosophy and said list are of course confined to the present context of the model algorithm for range finding. It however remains to be explored how good this philosophy holds in other kinds of algorithms, such as interval Newton methods and interval global optimization.

In conclusion, we believe that the findings of the present work indicate that it may be fruitful to bring in some amount of randomization into the body of interval analysis algorithms.

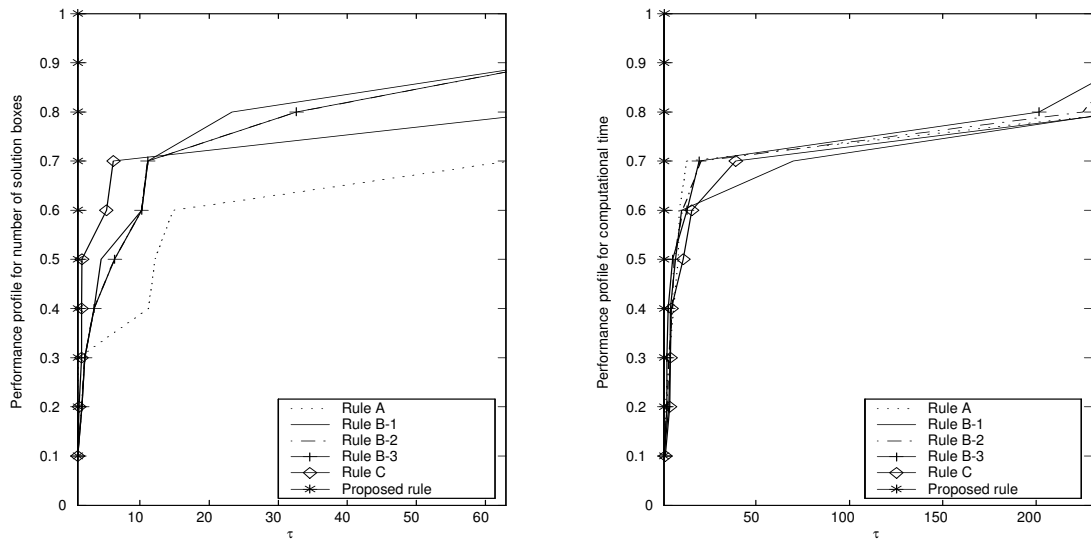


FIGURE 6.1. Performance profile plots for the number of solution boxes and computational time metrics.

TABLE 6.1. Performance of the proposed bisection rule over different runs.

Sl. No.	Appl. Example	$l$	Solutions	Run 1	Run 2	Run 3	Run 4	Run 5	Average
1	Active Noise & Vibration	2	boxes	28, 236	27, 552	27, 508	27, 309	26, 645	27, 450
			time (s)	2.86	2.77	2.74	2.77	2.7	2.77
2	Simple poles	3	boxes	459	459	617	617	459	522
			time (s)	0.37	0.36	0.39	0.39	0.39	0.38
3	NMP	3	boxes	504	512	512	512	504	509
			time (s)	0.36	0.36	0.33	0.33	0.35	0.35
4	Non-rational	3	boxes	6759	6727	7527	6727	6759	6900
			time (s)	2.59	2.4	2.52	2.37	2.38	2.45
5	Vehicle clutch	3	boxes	2, 499	2, 502	2, 458	2, 499	2, 543	2, 500
			time (s)	1.04	1.02	1.02	0.98	1	1
6	Electro - mechanical	5	boxes	84, 273	84, 740	89, 628	84, 308	84, 649	85, 220
			time (s)	11.83	11.89	12.64	11.86	11.88	12.02
7	Mechanical	5	boxes	1666	1557	1335	1309	1811	1536
			time (s)	0.95	0.84	0.79	0.79	0.76	0.83
8	Aircraft	5	boxes	605	688	640	640	570	629
			time (s)	0.74	0.79	0.78	0.75	0.71	1.15
9	DC Motor	6	boxes	6236	6231	6236	6231	6236	6234
			time (s)	1.19	1.15	1.14	1.14	1.13	1.15
10	Inv. Pendulum	7	boxes	208	194	194	194	194	197
			time (s)	0.54	0.46	0.46	0.47	0.47	0.48



TABLE 6.2. Comparison of the 'best bisection direction'  $k$  for the random box versus the actual 'best bisection direction' found for all other boxes in list over various iterations.  $\alpha$  in Column 5 denotes the number of iterations where more than 90 percent of boxes have the same 'best bisection direction' as the random box.

Sl. No.	Application Example	$l$	Number of iterations	$\alpha$	Average % of boxes having same $k$ as random box
1	Active Noise	2	18	4	70%
2	Simple poles	3	11	5	80%
3	NMP	3	9	6	88%
4	Non rational	3	14	6	75%
5	Vehicle clutch	3	16	8	79%
6	Electro - Mech.	5	21	7	73%
7	Mechanical	5	15	6	74%
8	Aircraft	5	13	5	74%
9	DC Motor	6	14	7	73%
10	Inv. Pendulum	7	8	6	89%

TABLE 6.3. Number of solution boxes obtained with different bisection rules.

Sl. No.	Application Example	$l$	Rule A	Rule B-1	Rule B-2	Rule B-3	Rule C	Proposed Rule
1	Active Noise	2	30,223	280,657	280,657	280,657	28,181	27450
2	Simple poles	3	7,712	521	521	521	627	522
3	NMP	3	512	808	808	808	808	509
4	Non-rational	3	6,759	13,673	13,673	13,673	35,404	6,900
5	Vehicle clutch	3	30,424	*	*	*	3865	2500
6	Electro - Mech.	5	*	290,594	290,594	290,594	*	85,220
7	Mechanical	5	17,320	17,107	17,107	17,107	*	1,536
8	Aircraft	5	40,002	2,752	4,007	4,007	966	629
9	DC Motor	6	*	145,302	203,663	203,663	38,480	6,234
10	Inv. Pendulum	7	*	*	*	*	*	197

TABLE 6.4. Computational time taken with different bisection rules.

Sl. No.	Application Example	$l$	Rule A	Rule B-1	Rule B-2	Rule B-3	Rule C	Proposed Rule
1	Active Noise	2	5.24	194.28	57.93	54.54	109	2.77
2	Simple poles	3	3.27	1.60	2.44	0.91	1.88	0.38
3	NMP	3	1.33	1.57	0.92	0.88	1.59	0.35
4	Non rational	3	3.12	11.28	12.41	9.8	35.23	3.55
5	Vehicle clutch	3	6.09	*	*	*	4.09	1
6	Electro - Mech.	5	*	125	132.14	158.75	*	12.02
7	Mechanical	5	10.89	5.85	4.55	4.58	*	0.83
8	Aircraft	5	10.73	1.16	1.31	1.32	1.92	1.15
9	DC Motor	6	*	282.42	258.47	231.66	18.18	1.15
10	Inv. Pendulum	7	*	*	*	*	*	0.48

TABLE 6.5. Ranking of rules based on the number of solution boxes.

Sl. No.	Rules	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6
1	Rule A	1	1	2	2	1	3
2	Rule B-1	1	3	4	—	—	2
3	Rule B-2	1	3	2	2	—	2
4	Rule B-3	1	3	2	2	—	2
5	Rule C	—	3	3	1	—	3
6	Proposed	8	2	—	—	—	—

TABLE 6.6. Ranking of rules based on the computational time taken.

Sl. No.	Rules	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6
1	Rule A	1	1	1	1	1	5
2	Rule B-1	—	2	1	2	2	3
3	Rule B-2	—	1	3	2	2	2
4	Rule B-3	—	2	4	2	—	2
5	Rule C	—	2	0	1	2	5
6	Proposed	9	1	—	—	—	—

TABLE 6.7. Percentage reduction obtained with the proposed rule over existing rules.

Sl. No.	Appl. Example	$l$	Solutions	Rule A	Rule B-1	Rule B-2	Rule B-3	Rule C
1	Active Noise & Vibration	2	boxes	9%	90%	90%	90%	3%
			time (s)	47%	99%	95%	95%	97%
2	Simple poles	3	boxes	93%	0%	0%	0%	17%
			time (s)	88%	76%	84%	58%	79%
3	NMP	3	boxes	1%	37%	37%	37%	37%
			time (s)	74%	78%	62%	60%	78%
4	Non-rational	3	boxes	-2%	50%	50%	50%	81%
			time (s)	-14%	69%	71%	64%	89%
5	Vehicle clutch	3	boxes	92%	*	*	*	35%
			time (s)	84%	*	*	*	76%
6	Electro - Mechanical	5	boxes	*	71%	71%	71%	*
			time (s)	*	90%	91%	92%	*
7	Mechanical	5	boxes	91%	91%	91%	91%	91%
			time (s)	92%	86%	82%	82%	*
8	Aircraft	5	boxes	98%	77%	84%	84%	*
			time (s)	89%	1%	12%	13%	40%
9	DC Motor	6	boxes	*	95%	97%	97%	97%
			time (s)	*	100%	100%	100	94
10	Inv. Pendulum	7	boxes	*	*	*	*	*
			time (s)	*	*	*	*	*
	Average		boxes	55%	64%	65%	65%	42%
			time (s)	67%	75%	75%	71%	79%

# 7

## An algorithm for robust stability analysis

### 7.1 Introduction

Consider a family of real  $n$ th order polynomials

$$\{p(s, q), q \in \mathbf{Q}^0\} \quad (7.1)$$

where,  $s$  is the complex variable,  $q \in \mathbb{R}^l$  is a vector of physical parameters occurring possibly nonlinearly in the coefficients of polynomial  $p$ , and  $\mathbf{Q}^0 \subseteq \mathbb{R}^l$  is a bounding box for  $q$  given as

$$\mathbf{Q}^0 := \left( [\underline{\mathbf{Q}}_1^0, \overline{\mathbf{Q}}_1^0], \dots, [\underline{\mathbf{Q}}_l^0, \overline{\mathbf{Q}}_l^0] \right)$$

We wish to find if the above polynomial family is robustly stable, that is, if

$$p(s, q) \neq 0 \text{ for all } s \in C \text{ with } \operatorname{Re} s \geq 0, \text{ for all } q \in \mathbf{Q}^0 \quad (7.2)$$

Ackerman [1] and Barmish [6] proposed robust stability analysis methods for the case of the polynomial coefficients having affine or multilinear dependencies on the parameters  $q$ . More recently, Garloff and co-workers [19], [82] proposed Bernstein polynomial based robust stability analysis methods for the case of polynomial parametric dependencies. However, there seems to be a lack of robust stability analysis methods for the general case of nonlinear parametric dependencies.

In this work, we address this gap by presenting an algorithm to analyze the robust stability of polynomials with nonlinear parametric dependencies. We develop the proposed algorithm using tools of interval analysis. The proposed algorithm is applicable to nonlinear parametric dependencies of a very general class where the polynomial coefficients can be any continuous function of the parameters  $q$ .

## 7.2 Background

At the outset, we verify that at least one member of the polynomial family in (7.1) is stable. Otherwise, the family is obviously *not* robustly stable, and we may therefore exit the test.

Let  $\omega \in \Re$  denote the frequency variable. Expressing the polynomial  $p(j\omega, q)$  in terms of its even and odd parts gives

$$p(j\omega, q) = p_e(\omega^2, q) + j\omega p_o(\omega^2, q)$$

Now, at least one polynomial has already been verified as stable. Therefore, by the continuous dependency of the zeros of a polynomial on its coefficients, (7.2) implies that the entire polynomial family in (7.1) is robustly stable if and only if

$$0 \notin \{(p_e(\omega^2, q), p_o(\omega^2, q)), q \in \mathbf{Q}^0, \omega \in \Re\} \quad (7.3)$$

Define

$$\begin{aligned} x &= (\omega^2, q) \\ f(x) &= (p_e(x), p_o(x)) \end{aligned}$$

and let  $\mathbf{X}$  denote the interval vector corresponding to  $x$ , with  $\mathbf{X}^0 = \Re \times \mathbf{Q}^0$ . Then, we can write (7.3) as

$$0 \notin \{f(x), x \in \mathbf{X}^0\}$$

If  $F$  is any inclusion function of  $f$ , such as the natural inclusion function [64], then it follows from the inclusion property of interval analysis [49] that, for any  $\mathbf{X} \subseteq \mathbf{X}^0$ ,

$$0 \notin F(\mathbf{X}) \Rightarrow 0 \notin \{f(x), x \in \mathbf{X}\}$$

This is the basis for the interval zero exclusion test used below.

## 7.3 Proposed algorithm

An initial search box  $\mathbf{X}^0$  must be input to the proposed algorithm. For constructing this initial box, we may follow the method proposed by Garloff *et al.* [19] to obtain a tight initial frequency interval. Alternatively, we may use a ‘large’ initial frequency interval, but at the cost of increased computations. The parameter box is, of course, the given box  $\mathbf{Q}^0$ .

### Proposed algorithm

*Inputs:* An inclusion function  $F$  of  $f$ , the initial box  $\mathbf{X}^0 \in I(\Re^{l+1})$ .

*Output:* The message ‘the system is robustly stable’ or ‘the system is not robustly stable’.

BEGIN Algorithm

1. Set  $\mathbf{X} = \mathbf{X}^0$ , initialize list  $L = \{\mathbf{X}\}$ .

2. Remove all boxes from list  $L$  and evaluate  $F$  over all the boxes.
3. (interval zero exclusion test) Discard all boxes for which  $0 \notin F(\mathbf{X})$ .
4. For any box, if the width of  $F(\mathbf{X})$  is within machine precision, then print ‘the system is not robustly stable’ and EXIT algorithm.
5. Following section 6.3, bisect all boxes in coordinate direction  $k$ , getting subboxes  $\mathbf{V}^1, \mathbf{V}^2$  such that  $\mathbf{X} = \mathbf{V}^1 \cup \mathbf{V}^2$ . Deposit all these subboxes in  $L$ .
6. If the list  $L$  is empty, then print ‘the system is robustly stable’ and EXIT algorithm. Else, go to step 2.

END Algorithm

**Remark 7.1** *In the proposed algorithm, function evaluations, zero exclusion checks, width checks, and bisections on all boxes in an iteration are performed concurrently using vectorized interval arithmetic operations.*

**Remark 7.2** *In the proposed algorithm, we use the new bisection rule given in chapter 6.*

## 7.4 Application: mass-spring-damper

The example is given by Ackerman and Sienel [2]. Consider the mass-spring-damper system (also called as ‘plant’ in the sequel) shown in Fig. 7.1

The input  $u$  is a force accelerating mass  $m_2$ , whereas the measured variable is position  $y_1$ . Laplace transforming the equations of motion gives

$$\begin{aligned} p_1(s, m_1, d_1, c_1, c_{12}) x_1(s) - c_{12} x_2(s) &= 0 \\ p_2(s, m_2, d_2, c_2, c_{12}) x_2(s) - c_{12} x_1(s) &= u(s) \end{aligned}$$

where,

$$\begin{aligned} p_1(s, m_1, d_1, c_1, c_{12}) &= m_1 s^2 + d_1 s + c_1 + c_{12} \\ p_2(s, m_2, d_2, c_2, c_{12}) &= m_2 s^2 + d_2 s + c_2 + c_{12} \end{aligned}$$

The values of the parameters vary over the intervals

$$\begin{aligned} m_1 &\in [1, 3] & d_1 &\in [0.5, 2] & c_1 &\in [1, 2] \\ m_2 &\in [2, 5] & d_2 &\in [0.5, 2] & c_2 &\in [2, 4] \end{aligned}$$

with fixed  $c_{12} = 1$ . The ‘nominal’ plant is the one corresponding to the midpoint of each parameter interval. Using the pole-placement technique for the nominal plant, a minimum-phase and stable third order compensator of the form

$$\frac{n_c(s, b)}{d_c(s, a)} = \frac{b_3 s^3 + b_2 s^2 + b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s + a_0}$$

is designed. Now a neighborhood of this compensator with  $\pm 10\%$  variation in the coefficients is assumed, giving

$$\begin{aligned} a_0 &\in [17100, 20900] & a_1 &\in [1305, 1595] & a_2 &\in [55.8, 68.2] \\ b_0 &\in [212062.5, 259187.5] & b_1 &\in [805837.5, 984912.5] & b_2 &\in [721012.5, 881237.5] \\ b_3 &\in [424125, 518375] \end{aligned}$$

The feedback system has the closed-loop characteristic polynomial

$$\{p_1(s, c_1, d_1, m_1, c_{12})p_2(s, c_2, d_2, m_2, c_{12}) - 1\}d_c(s, a) + n_c(s, b)$$

having thirteen uncertain parameters, and with multilinear parametric dependencies. The initial frequency interval is taken as  $[0, 5000]$ . Note that Garloff *et al.* [19] give the initial frequency interval as  $[0.48, 4879.76]$ .

The proposed algorithm is implemented on a Pentium III 800 MHz machine with 500 MB RAM using INTLAB [68]. The algorithm reports in 0.14 seconds that the considered feedback system is robustly stable.

It may be noted that for the same example, Garloff *et al.* [19] report that their Convex Hull Bernstein algorithm took about 80 seconds on a more powerful HP 9000/755 workstation. Moreover, the method of Garloff *et al.* is restricted to parametric dependencies which are polynomial in nature, whereas the proposed algorithm is applicable to parametric dependencies of a general nonlinear nature.

## 7.5 Conclusions

We have presented a simple and direct method for robust stability analysis of polynomials having nonlinear parametric dependencies. We successfully demonstrated the effectiveness of the proposed algorithm on a physical example with thirteen uncertain parameters. By considering the difficult and general class of nonlinear parametric dependencies, we believe that the proposed method fills in a significant gap in the robust stability analysis literature.

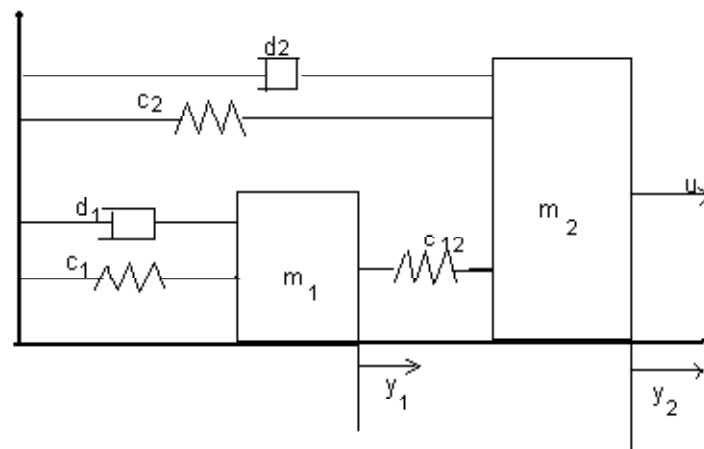


FIGURE 7.1. Schematic representation of the mass-spring-damper system.





# 8

## Conclusions

Although interval analysis algorithms are reliable and accurate, they are often found to be slow in execution, specially for difficult problems. The speed limitation has been overcome to a considerable extent by using special coprocessors, functional hardware units, and parallel processing techniques on multi-processor machines. However, as single processor machines such as desktop computers are much more widely used, it would be advantageous to have techniques that speed up interval analysis algorithms on single processor machines.

In this thesis, an attempt was made to speed up some of the key interval analysis algorithms on *single* processor machines. In a typical interval analysis algorithm, only one box is processed in each iteration. Since such processing is inherently slow due to its *sequential* nature, it was therefore proposed in this work to process *all* boxes present in each iteration. Further, it was also proposed to use *vectorized* interval arithmetic operations for performing the various tasks, such as function and gradient evaluations, monotonicity test, cut-off test, width checks, and bisections, on all boxes present at a given iteration. Vectorization gives a degree of *parallelism* in scalar processors, enabling faster processing of all boxes in the list at a given iteration. This kind of vectorization or data parallelism is currently possible with INTLAB/MATLAB, FORTRAN 95, C / C++, and PASCAL for interval arithmetic operations.

The proposed strategy<sup>1</sup> led to the following developments and consequences:

1. A vectorized version of the interval Newton algorithm was proposed for solving finite-dimensional systems of nonlinear equations. The proposed vectorized interval Newton algorithm is unaltered in essence from the basic interval Newton algorithm. Yet, on a collection of twenty-one test problems, the proposed vectorized algorithm was found to

---

<sup>1</sup>With other algorithmic changes, noted wherever applicable.

be considerably faster than the original one in *all* examples. On an average, the former was found to be more than *eleven* times faster than the latter.

2. A vectorized version of Neumaier's covering algorithm was proposed for solving finite-dimensional systems of parameter - dependent nonlinear equations. The proposed vectorized covering algorithm is unaltered in essence from the basic covering algorithm. On a collection of five test examples, the proposed vectorized covering algorithm was found to be significantly faster than the original one in *all* examples. In demanding problems, the former was found to be faster by up to *two* orders of magnitude than the latter. Moreover, the speed up with the proposed vectorized version was found to get better with tighter prescribed accuracy.
3. A vectorized version of the basic interval analysis optimization algorithm was proposed for solving the unconstrained nonlinear global optimization problem. The basic optimization algorithm referred to is the well-known Moore-Skelboe algorithm augmented with the midpoint test and the monotonicity test. Algorithmic changes to the basic algorithm were also proposed, based on the fact that all the boxes in a given list were processed. In the basic algorithm, the cut-off test uses the function value at the midpoint of the *leading* box at each iteration, while the monotonicity test is applied to the subboxes obtained by bisection of the *leading* box at each iteration. In the proposed algorithm, the cut-off test uses the minimum of the function values at the midpoints of *all* boxes in the list at each iteration, while the monotonicity test is applied to the subboxes obtained by bisection of *all* boxes in the list at each iteration. The new cut-off test in the proposed algorithm is more effective than the one in the basic algorithm - as the new cut-off value is the minimum of function values at the midpoints of all boxes in the list. Likewise, the new monotonicity test in the proposed algorithm enables irrelevant regions to be discarded sooner than in the basic algorithm. The proposed and existing algorithms were evaluated using two different termination criteria. On a collection of fifty standard optimization test functions, the proposed algorithm was able to solve *all* the test functions, whereas the basic algorithm was unable to do so. Despite more number of function evaluations, computational effort, and maximum list length, the proposed algorithm gave an average speed improvement of 68% for termination criterion A. For termination criterion B, the above performance metrics were more favorable for the proposed algorithm with the average speed improvement being 74%. In short, the proposed algorithm used any extra function evaluations, computational effort, and maximum list length effectively, yielding considerable savings in computational time.
4. A vectorized version of the set inversion via interval analysis algorithm of Jaulin *et al.* was proposed for characterizing the domain of a set of nonlinear inequalities. Algorithmic improvements to the algorithm of Jaulin *et al.* were also proposed, based on the

property of monotonicity: (a) the powerful monotonicity test form was used as an inclusion function, and (b) when any of the constraint functions monotonically increased resp. decreased in every component direction on a given box, then the part of box where the inequality was certainly infeasible was found and discarded. On two robust stability problems, including a case study of speed control of a jet engine, it was found that the proposed algorithm enclosed  $\mathcal{S}$  more accurately. The proposed algorithm was also found to require less list lengths, but at the expense of more computational time.

5. A derivative free rule for bisection direction selection was proposed for use in an existing vectorized algorithm for range computations. The philosophy behind the proposed rule was that “the best bisection direction for the random box of current list is (likely) the best bisection direction for all other boxes of current list”. On a collection of ten real-world examples concerning the template generation problem, it was found that the proposed bisection direction selection rule was able to successfully solve *all* examples, whereas *none* of the established rules, such as ‘maximum width’ and ‘maximum smear’ rules, were able to do so. Moreover, on the average, about 58% reduction in the number of solution boxes and 73% reduction in computational time were obtained with the proposed bisection direction selection rule over existing rules. A noteworthy empirical finding was that more than three-fourths of the boxes at any given iteration of the algorithm had the same ‘best bisection direction’ as that of the random box. This empirical finding vindicated, by and large, the philosophy adopted for the proposed rule “the best bisection direction for a randomly picked box from a given list is likely the best bisection direction for all other boxes of the same list”. The findings of the present work also suggest that it may be fruitful to bring in some amount of *randomization* into the body of interval analysis algorithms.
6. A vectorized interval analysis algorithm was proposed to analyze the robust stability of polynomials with nonlinear parametric dependencies. The proposed algorithm is applicable to nonlinear parametric dependencies of a very general class where the polynomial coefficients can be any continuous function of the parameters. The proposed algorithm was based on the interval zero exclusion test, and used the above proposed derivative free rule for bisection of all boxes in a given list. On a mass-spring-damper system involving thirteen uncertain physical parameters, it was found that the algorithm successfully verified robust stability. As there is currently a lack of robust stability analysis methods for the general case of nonlinear parametric dependencies, it is believed that the proposed method fills in a significant gap in the robust stability literature.

## 8.1 Suggestions for future work

Some suggested directions for future research work are as follows:

- Investigation of a more comprehensive vectorized global optimization method that includes accelerating devices, such as the convexity test and the interval Newton method.
- Extension of the above to deal with *constrained* global optimization problems.
- Investigation of a randomized bisection direction rule in the context of global optimization and root finding.
- Investigation of an improved set inversion algorithm using Hansen's strategy [26] for solving system of inequalities .
- Development of application specific hardware, such as those based on floating - point gate arrays (FPGAs), for performing vectorized interval Newton algorithm and Moore - Skelboe algorithm.

# Appendix A

## Test problems used in chapter 2

1. **Toolbox1.** Zeros of this function correspond to the intersection points of the circle with mid point  $(10, 1)$  and radius 1 and the circle with mid point  $(11, 1)$  and radius 1, The function is given as Exercise 13.1 in [24].

$$\begin{aligned}
 f_1 &= x_1^2 - 20x_1 + x_2^2 - 2x_2 + 100 \\
 f_2 &= x_1^2 - 22x_1 + x_2^2 - 2x_2 + 121
 \end{aligned}$$

The initial box is  $([10, 12], [0, 3])$

2. **Toolbox2.** The function is given as Exercise 13.2 in [24].

$$\begin{aligned}
 f_1 &= \left(\frac{x_1}{2}\right)^2 + \left(\frac{x_2}{3}\right)^2 - 1 \\
 f_2 &= \left(\frac{x_1}{3}\right)^2 + \left(\frac{x_2}{2}\right)^2 - 1
 \end{aligned}$$

The initial box is  $([1, 2], [1, 2])$

3. **Hansen1.** A numerical example given in [26, pp. 110].

$$\begin{aligned}
 f_1 &= x_1^2 + x_2^2 - 1 \\
 f_2 &= 2x_1^2 - x_2 - 1
 \end{aligned}$$

The initial box is  $([0, 2], [-2, 2])$

4. **Lotka3.** A neural network modeled by an adaptive Lotka-Volterra system (given as problem noon3 in [78]) .

$$\begin{aligned}
 f_1 &= x_1x_2^2 + x_1x_3^2 - 1.1x_1 + 1 \\
 f_2 &= x_2x_1^2 + x_2x_3^2 - 1.1x_2 + 1 \\
 f_3 &= x_3x_1^2 + x_3x_2^2 - 1.1x_3 + 1
 \end{aligned}$$

The initial box is  $[-1.5, 2]^3$

5. **Rediff3**. A 3–dimensional reaction-diffusion problem in [78].

$$\begin{aligned} f_1 &= -2x_1 + x_2 + 0.835634534x_1(1 - x_1) \\ f_2 &= x_1 - 2x_2 + x_3 + 0.835634534x_2(1 - x_2) \\ f_3 &= x_2 - 2x_3 + 0.835634534x_3(1 - x_3) \end{aligned}$$

The initial box is  $[0, 1]^3$

6. **Redcyc4**. A reduced cyclic 4–roots problem in [78].

$$\begin{aligned} f_1 &= 1 + x_1 + x_2 + x_3 + x_4 \\ f_2 &= x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4 \\ f_3 &= x_1x_2 + x_1x_2x_3 + x_2x_3x_4 + x_2x_3x_4 + x_3x_4 + x_4x_1 \\ f_4 &= x_1x_2x_3 + x_1x_2x_3x_4 + x_2x_3x_4 + x_3x_4x_1 + x_4x_1x_2 \end{aligned}$$

The initial box is  $([-0.4, 6.9], [-2.7, 1], [-2.7, -0.2], [-2.7, 1])$

7. **Lorentz4**. The equilibrium points of a 4–dimensional Lorentz attractor in [78].

$$\begin{aligned} f_1 &= x_1x_2 - x_1x_3 - x_4 + 1 \\ f_2 &= x_2x_3 - x_2x_4 - x_1 + 1 \\ f_3 &= -x_1x_3 + x_3x_4 - x_2 + 1 \\ f_4 &= x_1x_4 - x_2x_4 - x_3 + 1 \end{aligned}$$

The initial box is  $([-1, 1], [-0.1, 1], [-1, 1], [-0.1, 1])$

8. **Quad4**. A Gaussian quadrature formula with 2 knots and 2 weights (given as problem quadfor in [78]).

$$\begin{aligned} f_1 &= x_1 + x_2 - 1 \\ f_2 &= x_1x_3 + x_2x_4 \\ f_3 &= x_1x_3^2 + x_2x_4^2 - 2/3 \\ f_4 &= x_1x_3^3 + x_2x_4^3 \end{aligned}$$

The initial box is  $([-1, 1], [0, 1], [0, 1], [-2, 0])$

9. **Lotka5**. A neural network modeled by an adaptive Lotka-Volterra system (given as problem noon5 in [78]).

$$f_1 = x_1x_2^2 + x_1x_3^2 + x_1x_4^2 + x_1x_5^2 - 1.1x_1 + 1$$

$$\begin{aligned}
 f_2 &= x_2x_1^2 + x_2x_3^2 + x_2x_4^2 + x_2x_5^2 - 1.1x_2 + 1 \\
 f_3 &= x_3x_1^2 + x_3x_2^2 + x_3x_4^2 + x_3x_5^2 - 1.1x_3 + 1 \\
 f_4 &= x_4x_1^2 + x_4x_2^2 + x_4x_3^2 + x_4x_5^2 - 1.1x_4 + 1 \\
 f_5 &= x_5x_1^2 + x_5x_2^2 + x_5x_3^2 + x_5x_4^2 - 1.1x_5 + 1
 \end{aligned}$$

Initial box :  $([-0.9, -0.3], [-0.8, -0.3], [-0.8, -0.3], [-0.8, -0.3], [-0.8, 2])$

10. **Eco5**. A 5-dimensional Economics problem in [78].

$$\begin{aligned}
 f_1 &= (x_1 + x_1x_2 + x_2x_3 + x_3x_4)x_5 - 1 \\
 f_2 &= (x_2 + x_1x_3 + x_2x_4)x_5 - 2 \\
 f_3 &= (x_3 + x_1x_4)x_5 - 3 \\
 f_4 &= x_1 + x_2 + x_3 + x_4 + 1
 \end{aligned}$$

The initial box is  $([-0.1, 1.3], [0.9, 1.7], [-1.2, 1.2], [-5, -1], [-10, -0.9])$

11. **Wright5**. The system of A. H. Wright in [78].

$$\begin{aligned}
 f_1 &= x_1^2 - x_1 + x_2 + x_3 + x_4 + x_5 - 10 \\
 f_2 &= x_2^2 + x_1 - x_2 + x_3 + x_4 + x_5 - 10 \\
 f_3 &= x_3^2 + x_1 + x_2 - x_3 + x_4 + x_5 - 10 \\
 f_4 &= x_4^2 + x_1 + x_2 + x_3 - x_4 + x_5 - 10 \\
 f_5 &= x_5^2 + x_1 + x_2 + x_3 + x_4 - x_5 - 10
 \end{aligned}$$

The initial box is  $([-5, 5.4], [-5, 4], [-5, 2], [-5, 3.1], [-5, 2.4])$

12. **Redeco5**. A reduced 5-dimensional Economics problem in [78]

$$\begin{aligned}
 f_1 &= x_1 + x_1x_2 + x_2x_3 + x_3x_4 - u_5 \\
 f_2 &= x_2 + x_1x_3 + x_2x_4 - 2u_5 \\
 f_3 &= x_3 + x_1x_4 - 3u_5 \\
 f_4 &= x_4 - 4u_5 \\
 f_5 &= x_1 + x_2 + x_3 + x_4 + 1, \text{ where } u_5 = 1/x_5
 \end{aligned}$$

The initial box is  $([-0.3, 1], [-1.5, 1.7], [-1.2, 1], [-4, 0], [-1, 0])$ .

13. **Neuro6**. A problem from Neurophysiology (given as problem boon in [78]).

$$\begin{aligned}
 f_1 &= x_1^2 + x_3^2 - 1 \\
 f_2 &= x_2^2 + x_4^2 - 1
 \end{aligned}$$



$$\begin{aligned}
f_3 &= x_5x_3^3 + x_6x_4^3 - 1.2 \\
f_4 &= x_5x_1^3 + x_6x_2^3 - 1.2 \\
f_5 &= x_5x_3^2x_1 + x_6x_4^3x_2 - 0.7 \\
f_6 &= x_5x_3x_1^2 + x_6x_4x_2^2 - 0.7
\end{aligned}$$

The initial box is  $([-0.5, 0], [0, 1], [-1, 1], [-1, 1], [-1, 1], [-0.5, 2])$ .

14. **Trink.** A system of Trinks from the PoSSo test suite, also given in [78].

$$\begin{aligned}
f_1 &= 45x_1 + 35x_2 - 165x_3 - 36 \\
f_2 &= 35x_1 + 25x_4 + 40x_5 - 27x_2 \\
f_3 &= 25x_1x_2 - 165x_3^2 + 15x_6 - 18x_4 + 30x_5 \\
f_4 &= 15x_1x_4 + 20x_5x_2 - 9x_6 \\
f_5 &= -11x_3^3 + x_6x_1 + 2x_4x_5 \\
f_6 &= -11x_2x_3 + 3x_3^2 + 99x_6
\end{aligned}$$

The initial box is  $([-3.5, 0.3], [-3.5, 0.4], [-1.9, 0], [-7, 0.1], [-0.1, 5], [-0.1, 0.8])$ .

15. **Redeco6.** A reduced 6-dimensional Economics problem in [78].

$$\begin{aligned}
f_1 &= x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 - u_6 \\
f_2 &= x_2 + x_1x_3 + x_2x_4 + x_3x_5 - 2u_6 \\
f_3 &= x_3 + x_1x_4 + x_2x_5 - 3u_6 \\
f_4 &= x_4 + x_1x_5 - 4u_6 \\
f_5 &= x_5 - 5u_6 \\
f_6 &= x_1 + x_2 + x_3 + x_4 + x_5 + 1, \text{ where } u_6 = 1/x_6
\end{aligned}$$

The initial box is  $([-1, 1.3], [-1, 1.4], [-1.6, 1], [-1.2, 1.1], [-5, 0], [-2, 0])$ .

16. **Hansens.** The problem of Hansen and Sengupta, also worked out as a sample problem in [24].

$$f_i = 0.6x_i - 2 + 0.49x_i \sum_{j=1}^6 x_j^2, i = 1, \dots, 6$$

The initial box is  $[-1, 1]^6$ .

17. **Systems8.** A small system from constructive Galois theory (given as problem *s9.1* in [78]).

$$\begin{aligned}
f_1 &= -x_1x_7 - 2x_2x_3 \\
f_2 &= 9x_1 + 4x_4
\end{aligned}$$

$$\begin{aligned}
f_3 &= -4x_5x_3 - 2x_1x_6 - 3x_2x_7 \\
f_4 &= -7x_5 + 9x_8 - 8x_6 \\
f_5 &= -4x_2x_6 - 5x_5x_7 - 6x_3 - 3x_1 \\
f_6 &= -5x_2 - 6x_5x_6 - 7x_7 + 9x_4 \\
f_7 &= 9x_2 + 6x_8 - 5x_4 \\
f_8 &= 9x_5 - 7x_8 + 8
\end{aligned}$$

The initial box is  $([-3, 0.5], [-0.5, 1.2], [-0.5, 1.5], [-0.5, 6], [-2, 3], [0, 3], [0, 2, 2], [-1.5, 5])$ .

18. **Robot.** The kinematic equations for a PUMA robot in [53].

$$\begin{aligned}
f_1 &= \gamma_1 x_1x_3 + \gamma_2x_2x_3 + \gamma_3x_1 + \gamma_4x_2 + \gamma_5x_4 + \gamma_6x_7 + \gamma_7 = 0 \\
f_2 &= \gamma_8 x_1x_3 + \gamma_9x_2x_3 + \gamma_{10}x_1 + \gamma_{11}x_2 + \gamma_{12}x_4 + \gamma_{13} = 0 \\
f_3 &= \gamma_{14} x_6x_8 + \gamma_{15}x_1 + \gamma_{16}x_2 = 0 \\
f_4 &= \gamma_{17}x_1 + \gamma_{18} x_2 + \gamma_{19} = 0 \\
f_5 &= x_1^2 + x_2^2 - 1 = 0 \\
f_6 &= x_3^2 + x_4^2 - 1 = 0 \\
f_7 &= x_5^2 + x_6^2 - 1 = 0 \\
f_8 &= x_7^2 + x_8^2 - 1 = 0
\end{aligned}$$

where,

$$\begin{aligned}
\gamma_1 &= 4.731 \times 10^{-3}, \gamma_2 = -0.3578, \gamma_3 = -0.1238, \gamma_4 = -1.637 \times 10^{-3}, \gamma_5 = -0.9338, \gamma_6 = 1, \\
\gamma_7 &= -0.3571, \gamma_8 = 0.2238, \gamma_9 = 0.7623, \gamma_{10} = 0.2638, \gamma_{11} = -0.07745, \gamma_{12} = -0.6734, \\
\gamma_{13} &= -0.6022, \gamma_{14} = 1, \gamma_{15} = 0.3578, \gamma_{16} = 4.731 \times 10^{-3}, \gamma_{17} = -0.7623, \\
\gamma_{18} &= 0.2238, \gamma_{19} = 0.3461
\end{aligned}$$

The initial box is  $[-1, 1]^8$ .

19. **Kinema.** A robot kinematics problem in [78].

$$\begin{aligned}
f_1 &= x_1^2 + x_2^2 + x_3^2 - 12x_1 - 68 \\
f_2 &= x_4^2 + x_5^2 + x_6^2 - 12x_5 - 68 \\
f_3 &= x_7^2 + x_8^2 + x_9^2 - 24x_8 - 12x_9 + 100 \\
f_4 &= x_1x_4 + x_2x_5 + x_3x_6 - 6x_1 - 6x_5 - 52 \\
f_5 &= x_1x_7 + x_2x_8 + x_3x_9 - 6x_1 - 12x_8 - 6x_9 + 64 \\
f_6 &= x_4x_7 + x_5x_8 + x_6x_9 - 6x_5 - 12x_8 - 6x_9 + 32 \\
f_7 &= 2x_2 + 2x_3 - x_4 - x_5 - 2x_6 - x_7 - x_9 + 18
\end{aligned}$$

$$f_8 = x_1 + x_2 + 2x_3 + 2x_4 + 2x_6 - 2x_7 + x_8 - x_9 - 38$$

$$f_9 = x_1 + x_3 - 2x_4 + x_5 - x_6 + 2x_7 + 2x_8 + 8$$

The initial box is  $([2, 10], [-4, 10], [2, 10], [5, 10], [0, 6], [6, 7], [0, 10], [3, 11], [6, 11])$ .

20. **Ku10**. A 10-dimensional system of Ku in [78].

$$f_1 = 5x_1x_2 + 5x_1 + 3x_2 + 55$$

$$f_2 = 7x_2x_3 + 9x_2 + 9x_3 + 19$$

$$f_3 = 3x_3x_4 + 6x_3 + 5x_4 - 4$$

$$f_4 = 6x_4x_5 + 6x_4 + 7x_5 + 118$$

$$f_5 = x_5x_6 + 3x_5 + 9x_6 + 27$$

$$f_6 = 6x_6x_7 + 7x_6x_7 + 72$$

$$f_7 = 9x_7x_8 + 7x_7 + x_8 + 35$$

$$f_8 = 4x_8x_9 + 4x_8 + 6x_9 + 16$$

$$f_9 = 8x_9x_{10} + 4x_9 + 3x_{10} - 51$$

$$f_{10} = 3x_1x_{10} - 6x_1 + x_{10} + 5$$

The initial box is  $([2, 8], [-5, -2], [-2, 0], [1, 5], [-9, -4], [-4, -3], [2, 3], [-3, -2], [2, 4], [1, 2])$ .

21. **Sparsed1**. A sparse system known as benchmark D1 in [78].

$$f_1 = x_1^2 + x_2^2 - 1$$

$$f_2 = x_3^2 + x_4^2 - 1$$

$$f_3 = x_5^2 + x_6^2 - 1$$

$$f_4 = x_7^2 + x_8^2 - 1$$

$$f_5 = x_9^2 + x_{10}^2 - 1$$

$$f_6 = x_{11}^2 + x_{12}^2 - 1$$

$$f_7 = 3x_3 + 2x_5 + x_7 - 3.9701$$

$$f_8 = 3x_1x_4 + 2x_1x_6 + x_1x_8 - 1.7172$$

$$f_9 = 3x_2x_4 + 2x_2x_6 + x_2x_8 - 4.0616$$

$$f_{10} = x_3x_9 + x_5x_9 + x_7x_9 - 1.9791$$

$$f_{11} = x_2x_4x_9 + x_2x_6x_9 + x_2x_8x_9 + x_1x_{10} - 1.9115$$

$$f_{12} = -x_3x_{10}x_{11} - x_5x_{10}x_{11} - x_7x_{10}x_{11} + x_4x_{12} + x_6x_{12} + x_8x_{12} - 0.4077$$

The initial box is  $[-5, -0.1], [-1.1, -0.8], [0.4, 0.7], [-1.1, -0.699], [0.5, 1], [-1, -0.2], [0.5, 0.9], [-1, -0.2], [0.5, 1.1], [-0.5, 0.5], [0.5, 1.1], [-0.6, 0.2]$

# Appendix B

## Test problems used in chapter 3

1. The 1- dimensional manifold in [61]

$$x_1^3 - x_1x_2^2 + x_1^2 - x_1x_2 - x_2^2 = 0$$

where,  $\mathbf{X}_1 = [-3, 3]$ ,  $\mathbf{X}_2 = [-5, 5]$ .

2. This is an equation of a simple tunneling diode [76]

$$0.43x_1^3 - 2.69x_1^2 + 4.56x_1 = 2.5x_2^2 - 10.5x_2^2 + 11.8x_2 = i$$

where  $\mathbf{X}_1 = [0, 5]$ ,  $\mathbf{X}_2 = [0, 3]$ , and  $i = 5$  mA.

3. This problem is an example from combustion chemistry [53]. The system consists of two cubic equations:

$$\begin{aligned} \alpha_1 x_1^2 x_2 + \alpha_2 x_1^2 + \alpha_3 x_1 x_2 + \alpha_4 x_1 + \alpha_5 x_2 &= 0 \\ \alpha_6 x_1^2 x_2 + \alpha_7 x_1 x_2^2 + \alpha_8 x_1 x_2 + \alpha_9 x_2^3 + \alpha_{10} x_2^2 + \alpha_{11} x_2 + \alpha_{12} &= 0 \end{aligned}$$

where,  $\alpha_1 = -1.697 \times 10^7$ ,  $\alpha_2 = 2.177 \times 10^7$ ,  $\alpha_3 = 0.55$ ,  $\alpha_4 = 0.45$ ,  $\alpha_5 = -1$ ,  $\alpha_6 = 1.585 \times 10^{14}$ ,  $\alpha_7 = 4.126 \times 10^7$ ,  $\alpha_8 = -8.285 \times 10^6$ ,  $\alpha_9 = 2.284 \times 10^7$ ,  $\alpha_{10} = 1.918 \times 10^7$ ,  $\alpha_{11} = 48.4$ ,  $\alpha_{12} = -27.73$ . The box  $\mathbf{X}_1 = [0, 1]$ ,  $\mathbf{X}_2 = [0, 1]$ .

4. The hippopede problem in [44]

$$z = x_1^2 + x_2^2, \quad az = x_2^2 + z^2$$

where  $\mathbf{X}_1 = [-1.5, 1.5]$ ,  $\mathbf{X}_2 = [-1, 1]$ ,  $\mathbf{Z} = [0, 4]$ , and  $a = 1.1$ .

5. This is a set of kinematic equations for a PUMA robot in [53]

$$\begin{aligned}
\gamma_1 x_1 x_3 + \gamma_2 x_2 x_3 + \gamma_3 x_1 + \gamma_4 x_2 + \gamma_5 x_4 + \gamma_6 x_7 + \gamma_7 &= 0 \\
\gamma_8 x_1 x_3 + \gamma_9 x_2 x_3 + \gamma_{10} x_1 + \gamma_{11} x_2 + \gamma_{12} x_4 + \gamma_{13} &= 0 \\
\gamma_{14} x_6 x_8 + \gamma_{15} x_1 + \gamma_{16} x_2 &= 0 \\
\gamma_{17} x_1 + \gamma_{18} x_2 + \gamma_{19} &= 0 \\
x_1^2 + x_2^2 - 1 &= 0 \\
x_3^2 + x_4^2 - 1 &= 0 \\
x_5^2 + x_6^2 - 1 &= 0 \\
x_7^2 + x_8^2 - 1 &= 0
\end{aligned}$$

where,  $\gamma_1 = 4.731 \times 10^{-3}$ ,  $\gamma_2 = -0.3578$ ,  $\gamma_3 = -0.1238$ ,  $\gamma_4 = -1.637 \times 10^{-3}$ ,  $\gamma_5 = -0.9338$ ,  $\gamma_6 = 1$ ,  $\gamma_7 = -0.3571$ ,  $\gamma_8 = 0.2238$ ,  $\gamma_9 = 0.7623$ ,  $\gamma_{10} = 0.2638$ ,  $\gamma_{11} = -0.07745$ ,  $\gamma_{12} = -0.6734$ ,  $\gamma_{13} = -0.6022$ ,  $\gamma_{14} = 1$ ,  $\gamma_{15} = 0.3578$ ,  $\gamma_{16} = 4.731 \times 10^{-3}$ ,  $\gamma_{17} = -0.7623$ ,  $\gamma_{18} = 0.2238$ ,  $\gamma_{19} = 0.3461$ . The box  $\mathbf{X}_i \in [-1, 1]$ ,  $i = 1, \dots, 8$ .

# Appendix C

## Test problems used in chapter 6

For evaluating the performance of various bisection direction selection rules, we choose several challenging real-world application examples taken from different engineering disciplines. These are described below.

1. *Active noise and vibration control system* [67]: The phase angle and magnitude functions for a system occurring in active noise and vibration control with highly underdamped resonances are

$$\begin{aligned}
 f_1(x) &= -\frac{180}{\pi} \arctan \left\{ \frac{2x_2 \frac{\omega}{x_1}}{1 - \left( \frac{\omega}{x_1} \right)^2} \right\} \\
 f_2(x) &= -10 \log_{10} \left\{ \left( \left( \frac{\omega}{x_1} \right)^2 + 2x_2^2 - 1 \right)^2 + 1 - (2x_2^2 - 1)^2 \right\} \\
 x_1 &\in [0.75, 1.25], \quad x_2 \in [0.02, 0.06]
 \end{aligned}$$

The frequency is  $\omega = 1$ .

2. *Simple poles system* [56]: The phase angle and magnitude functions for a stable second order system with real poles are

$$\begin{aligned}
 f_1(x) &= -\frac{180}{\pi} \left\{ \arctan \left( \frac{\omega}{x_1} \right) + \arctan \left( \frac{\omega}{x_2} \right) \right\} \\
 f_2(x) &= -10 \log_{10} \{ (x_1^2 + x_2^2 + \omega^2) \omega^2 + (x_1 x_2)^2 \} + 20 \log_{10} (x_3) \\
 x_1 &\in [1, 5], \quad x_2 \in [20, 30], \quad x_3 \in [1, 10]
 \end{aligned}$$

The frequency is  $\omega = 1$ .

3. *Non-minimum phase system* [71]: The phase angle and magnitude functions for a non-minimum phase (NMP) system with real poles and zeros are

$$\begin{aligned} f_1(x) &= \frac{180}{\pi} \left( \arctan(-\omega x_2) - \arctan(\omega x_1) - \frac{\pi}{2} \right) \\ f_2(x) &= 10 \log_{10} \left\{ \frac{1 + (x_2 \omega)^2}{1 + (x_1 \omega)^2} \right\} + 20 \log_{10} \left( \frac{x_3}{\omega} \right) \\ x_1 &\in [0.3, 1], \quad x_2 \in [0.05, 0.1], \quad x_3 \in [1, 3] \end{aligned}$$

The frequency is  $\omega = 1$ .

4. *Non-rational system* [27, pp.129]: The phase angle and magnitude functions for a non-rational system are

$$\begin{aligned} f_1(x) &= -\frac{180}{\pi} \left( \arctan \left\{ \frac{-x_2 \sin(x_1 \omega)}{x_2 \cos(x_1 \omega) + 1} \right\} + \omega x_3 \right) \\ f_2(x) &= -10 \log_{10} \{1 + x_2(x_2 + 2 \cos(x_1 \omega))\} \\ x_1 &\in [1, 2], \quad x_2 \in [0.4, 0.6], \quad x_3 \in [0.01, 0.02] \end{aligned}$$

The frequency is  $\omega = 2$ .

5. *vehicle clutch system* [13]: The phase angle and magnitude functions between the input clutch position to the output transmission speed of a vehicle clutch system are

$$\begin{aligned} f_1(x) &= \frac{180}{\pi} \left( \arctan \left( \frac{24734.97\omega}{65.61(x_1 - x_2 \omega^2)} \right) - \arctan \left\{ \frac{3.07}{-1157.39\omega} \left( x_1 - \frac{\omega^2}{\left( \frac{1}{65.61} + \frac{1}{x_2} \right)} \right) \right\} \right) \\ f_2(x) &= 10 \log_{10} \left\{ \frac{(65.61(x_1 - x_2 \omega^2))^2 + (24734.97\omega)^2}{(-1157.39\omega)^2 + \left( 3.07 \left( x_1 - \frac{\omega^2}{\left( \frac{1}{65.61} + \frac{1}{x_2} \right)} \right) \right)^2} \right\} + \\ &\quad 20 \log_{10} \left( \frac{x_3}{(65.61 + x_2)\omega} \right) \\ x_1 &\in [5800, 115000], \quad x_2 \in [1400, 11000], \quad x_3 \in [100, 800] \end{aligned}$$

The frequency is  $\omega = 10$ .

6. *Electro - mechanical system* [14]: The phase angle and magnitude functions of an electro - mechanical system are

$$f_1(x) = \frac{180}{\pi} \left( \arctan \left( \frac{x_2 \omega}{x_3 - x_1 \omega^2} \right) - \arctan \frac{(x_1 + 0.4)x_3 - 0.4x_1 \omega}{(-x_1 + 0.4)x_2 \omega} \right)$$

$$f_2(x) = 10 \log_{10} \left( (x_2 \omega)^2 + (x_3 - x_1 \omega^2)^2 \right) - 10 \log_{10} \left( \frac{((-x_1 + 0.4) x_2 \omega^2)^2 + (\omega (x_1 + 0.4) x_3 - 0.4 x_1 \omega)^2}{(\omega (x_1 + 0.4) x_3 - 0.4 x_1 \omega)^2} \right)$$

$$x_1 \in [5.6, 8], x_2 \in [30, 300], x_3 \in [5880, 5900].$$

The frequency is  $\omega = 10\pi$ .

7. *Mechanical system* [27, pp. 222]: The phase angle and magnitude functions for a mechanical system are

$$f_1(x) = -\frac{180}{\pi} \left( \arctan \left\{ \frac{x_3}{\left( \frac{x_4}{\omega x_2} - x_1 \omega x_2 \right)} \right\} + \frac{\pi}{2} \right)$$

$$f_2(x) = -20 \log_{10} \left\{ \omega^2 \sqrt{x_3^2 + \omega x_2 \left( \frac{x_4}{(\omega x_2)^2} - x_1 \right)^2} \right\} + 20 \log_{10}(x_5)$$

$$x_1 \in [1, 2], x_2 \in [1, \sqrt{10}], x_3 \in [0.5, 1], x_4 \in [2, 3], x_5 \in [0.5, 2]$$

The frequency is  $\omega = 8$ .

8. *Aircraft system in longitudinal motion* [75]: The phase angle and magnitude functions for the longitudinal motion of an aircraft are

$$f_1(x) = \frac{180}{\pi} \left( \arctan \left( \frac{\omega}{x_1} \right) - \left\{ \arctan \left( \frac{\omega}{x_2} \right) + \frac{\pi}{2} + \arctan \left( \frac{2x_3 \frac{\omega}{x_4}}{1 - \left( \frac{\omega}{x_4} \right)^2} \right) \right\} \right)$$

$$f_2(x) = 10 \log_{10} \left\{ \frac{1 + \left( \frac{\omega}{x_1} \right)^2}{\omega^2 \left( 1 + \left( \frac{\omega}{x_2} \right)^2 \right) \left( \left( 1 - \left( \frac{\omega}{x_4} \right)^2 \right)^2 + \left( 2x_3 \frac{\omega}{x_4} \right)^2 \right)} \right\} + 20 \log_{10}(x_5)$$

$$x_1 \in [0.5, 0.75], x_2 \in [1, 10], x_3 \in [0.8, 0.9], x_4 \in [5, 6], x_5 \in [0.2, 2]$$

The frequency is  $\omega = 0.1$ .

9. *DC motor system* [5]: The phase angle and magnitude functions of a DC motor system are

$$f_1(x) = \frac{180}{\pi} \left( \arctan \left( \frac{x_1 x_2 \omega}{x_4} \right) + \arctan \left( \frac{x_6 (x_3 + x_4) + (x_5 (J_m + x_2))}{\omega^2 x_6 (-J_m - x_2) + x_5 (x_3 + x_4) + x_1^2} \right) \right)$$

$$f_2(x) = 10 \log_{10} \left( x_4^2 + (x_1 x_2)^2 \right) - 10 \log_{10} \left( \frac{(\omega^2 x_6 (-J_m - x_2) + x_5 (x_3 + x_4) + x_1^2)^2 + (x_6 (x_3 + x_4) + (x_5 (J_m + x_2)))^2}{(x_6 (x_3 + x_4) + (x_5 (J_m + x_2)))^2} \right)$$

$$x_1 \in [0.2, 0.6], x_2 \in [1e-5, 3e-5], x_3 = x_4 \in [1.95e-5, 2.05e-5],$$

$$x_5 \in [0.95, 1.05], x_6 \in [0.95e-2, 1.05e-2], J_m = 2e-3$$

The frequency is  $\omega = 20$ .



10. *Inverted pendulum system* [12]: The phase angle and magnitude functions between pendulum angle to the cart's motor current are

$$\begin{aligned}
 f_1(x) &= -\frac{180}{\pi} \left( \arctan \left\{ \frac{\omega}{- \left( x_2 x_1 + \frac{\omega^2}{x_4} \right)} \right\} + \arctan \left\{ \frac{2x_3 \frac{\omega}{x_5}}{1 - \frac{\omega^2}{x_5^2}} \right\} + \omega x_7 \right) \\
 f_2(x) &= 20 \log_{10} \left\{ \frac{\omega^2}{x_6 \omega^2 + 9.81} \frac{x_1}{\sqrt{\omega^2 + \left( x_2 x_1 + \frac{\omega^2}{x_4} \right)^2}} \frac{1}{\sqrt{\left( \frac{\omega}{x_5^2} + 2x_3^2 - 1 \right)^2 + 1 - (2x_3^2 - 1)^2}} \right\} \\
 x_1 &\in [1.5, 1.7], x_2 \in [0.05, 0.15], x_3 \in [0.01, 0.02] \\
 x_4 &\in [15, 17], x_5 \in [50, 60], x_6 \in [0.3, 0.45], x_7 \in [0.014, 0.015]
 \end{aligned}$$

The frequency is  $\omega = 1$ .

# Appendix D

## Interval analysis

Let  $\mathfrak{R}$  be the set of reals,  $x \in \mathfrak{R}^l$ ,  $\mathbf{X} = [\mathbf{X}_1, \dots, \mathbf{X}_l] \subseteq \mathfrak{R}^l$ , where  $\mathbf{X}_i = [\underline{\mathbf{X}}_i, \overline{\mathbf{X}}_i]$ ,  $i = 1, \dots, l$ .  $\mathbf{X}$  is an axis aligned parallelepiped also called as a box. Let  $I(\mathbf{X})$  be the set of all boxes contained in  $\mathbf{X}$ . Let the width of  $\mathbf{X}$  be defined as  $w(\mathbf{X}) = \max \mathbf{X} - \min \mathbf{X}$  if  $\mathbf{X} \in I(\mathfrak{R})$ , and as  $w(\mathbf{X}) = \max \{w(\mathbf{X}_1), \dots, w(\mathbf{X}_l)\}$ , if  $\mathbf{X} \in I(\mathfrak{R}^l)$ . Let the midpoint of  $\mathbf{X}$  be defined as  $m(\mathbf{X}) = (\min \mathbf{X} + \max \mathbf{X})/2$  if  $\mathbf{X} \in I(\mathfrak{R})$ , and as  $m(\mathbf{X}) = \{m(\mathbf{X}_1), \dots, m(\mathbf{X}_l)\}$ , if  $\mathbf{X} \in I(\mathfrak{R}^l)$ .

Let  $\bar{f}(\mathbf{X})$  denote the range of a function  $f$  on  $\mathbf{X}$ . A function  $F : I(\mathbf{X}) \rightarrow I(\mathfrak{R})$  is said to be an inclusion function [65] for  $f$ , if  $\bar{f}(\mathbf{Y}) \subseteq F(\mathbf{Y})$  for all  $\mathbf{Y} \in I(\mathbf{X})$ . A natural inclusion function form is obtained from the expression for  $f$  by replacing all occurrences of  $x_i$  with  $\mathbf{X}_i$  and all real operations with the corresponding interval operations.

Assume  $f$  to be continuously differentiable on  $\mathbf{X}$ , and let  $D_i F$  denote the natural inclusion function for the partial derivatives  $\partial f / \partial x_i$ ,  $i = 1(1)l$ . Then, the monotonicity test form  $F_{MT}$  [49] is an inclusion function form for  $f$ , and is given by

$$F_{MT}(\mathbf{X}) = [f(u), f(v)] + \sum_{i \in \mathcal{T}} D_i F(\mathbf{X})(\mathbf{X}_i - m_i), \quad (\text{D.1})$$

where,  $\mathcal{T}$  is the set of integers  $i$  such that  $D_i F(\mathbf{X}) = [\underline{D_i F(\mathbf{X})}, \overline{D_i F(\mathbf{X})}]$  properly contains zero, i.e.,  $\underline{D_i F(\mathbf{X})} < 0 < \overline{D_i F(\mathbf{X})}$ , and

$$(u_i, v_i) = \begin{cases} (\underline{\mathbf{X}}_i, \overline{\mathbf{X}}_i) & \text{if } \underline{D_i F(\mathbf{X})} \geq 0 \\ (\overline{\mathbf{X}}_i, \underline{\mathbf{X}}_i) & \text{if } \overline{D_i F(\mathbf{X})} \leq 0 \\ (m(\mathbf{X}_i), m(\mathbf{X}_i)) & \text{if } i \in \mathcal{T} \end{cases}$$

If  $\underline{D_i F(\mathbf{X})} \leq 0$  resp.  $\overline{D_i F(\mathbf{X})} \geq 0$  then  $f$  is monotonically decreasing resp. increasing in the  $i$ th component direction on  $\mathbf{X}$ . When  $\mathcal{T}$  becomes empty, it follows that  $f$  is monotonic on  $\mathbf{X}$  in all component directions  $i = 1(1)l$ .



## References

- [1] J. Ackermann. *Robust control: systems with uncertain physical parameters*. Springer-Verlag, 1975.
- [2] J. Ackermann and W. Sienel. What is a large number of parameters in robust systems. In *Proc. 29th IEEE Conf. on Decision and Control*, pages 3496–3497, 1990.
- [3] J. Ackermann and W. Sienel. On the computation of value sets for robust stability analysis. In *Proc. 1st European Control Conf.*, pages 1345–1350, Grenoble, France, 1991.
- [4] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, New York, 1983.
- [5] F. N. Bailey and D. Panzer. A fast algorithm for computing parametric rational functions. In *Proc. of ACC*, volume 1, pages 22–23, Atlanta, GA, USA, 1988.
- [6] B. R. Barmish. *New Tools for Robustness of Linear Systems*. McMillan, New York, USA, 1994.
- [7] S. Berner. Parallel methods for verified global optimization: practice and theory. *Journal of Global Optimization*, 9:1–22, 1996.
- [8] S. P. Bhattacharyya, H. Chapellat, and L. H. Keel. *Robust Control - The Parametric Approach*. Prentice Hall, New York, 1995.
- [9] A. Bik, M. Girkar, P. Grey, and X. Tian. Efficient exploitation of parallelism on Pentium III and Pentium 4 processor-based systems. *Intel Technology Q1*, pages 1–9, 2001.
- [10] A. Bik, M. Girkar, P. Grey, and X. Tian. Automatic intra-register vectorization for INTEL(R) architecture. *International Journal of Parallel Computing*, 30:65–98, 2002.

- [11] I. Bongartz, A. R. Conn, N. I. M. Gould, and M. A. Saunders. A numerical comparison between the LANCELOT and MINOS packages for large-scale numerical optimization. *Report 97/13 Namur University*, 1997.
- [12] C. Borghesani, Y. Chait, and O. Yaniv. The quantitative feedback theory toolbox for MATLAB. 1995.
- [13] W. Chen and D. J. Ballance. Plant template generation for uncertain plants in QFT. *Trans. of the ASME Journal of Dynamic Systems, Measurement and Control*, 121:359–364, 1999.
- [14] B. Cohen, M. Nordin, and P. O. Gutman. Recursive grid methods to compute value sets for transfer functions with parametric uncertainty. In *Proc. of ACC*, pages 3861–3865, 1995.
- [15] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM Jl. numerical analysis*, 34:922–938, 1997.
- [16] L. C. W. Dixon and M. Jha. Parallel algorithm for global optimization. *Journal of Optimization Theory and Applications*, 79:385–395, 1993.
- [17] E. D. Dolan and J. J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming Online*, October 2001.
- [18] J. J. Dongarra and V. Eijkhout. Numerical linear algebra algorithms and software. *Journal of Computational and Applied Mathematics*, 123:489–514, 2000.
- [19] J. Garloff, B. Graf, and M. Zettler. Speeding up an algorithm for checking robust stability of polynomials. In *Proc. 2nd IFAC Symposium on Robust Control Design*, pages 183–188. Elsevier Science, Oxford, 1998.
- [20] C. Gau and M. A. Stadtherr. Reliable nonlinear parameter estimation using an interval newton method: error-in-variable approach. *Computers and Chemical Engineering*, 24:631–638, 2000.
- [21] I. P. Grant and V. R. Saunders. Numerical computations of molecular integrals via optimized (vectorized) FORTRAN code. *Nuclear Instruments and Methods in Physics Research*, A 389:117–120, 1997.
- [22] J. W. Gudenberg. Hardware support for interval arithmetic. *Scientific Computing and Validated numerics*, pages 32–37, 1996.
- [23] U. L. Gunther, C. Ludwig, and H. Ruterjans. NMR LAB-Advanced NMR data processing in MATLAB. *Journal of Magnetic Resonance*, 145:201–208, 2000.

- [24] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *Numerical Toolbox for Verified Computing I*. Springer-Verlag, Berlin, 1993.
- [25] E. Hansen. Global optimization using interval analysis-The mutidimensional case. *Num. Mathematik*, 34:247–270, 1980.
- [26] E. Hansen. *Global optimization using interval analysis*. Marcel Dekker, 1992.
- [27] I. M. Horowitz. *Quantitative feedback design theory (QFT)*. QFT Publications, Boulder, Colorado, 1993.
- [28] J. Z. Hua, J. F. Brennecke, and M. A. Stadtherr. Reliable prediction of phase stability using an interval Newton method. *Fluid Phase Equilibria*, 116:52–59, 1996.
- [29] H. S. Huang and C. U. Lu. Vector implementations of fast decoupled and Gauss-Siedal load flows. *Electrical Power and Energy Systems*, 17(5):355–358, 1995.
- [30] K. Ichida and Y. Fujii. An interval arithmetic method for global optimization. *Computing*, 23:85–97, 1979.
- [31] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, London, 2001.
- [32] R. B. Kearfott. Abstract generalized bisection and a cost bound. *Mathematics of Computation*, 49(179):187–202, 1987.
- [33] R. B. Kearfott. Some tests of generalized bisection. *ACM Transactions on Mathematical Software*, 13(3):197–220, 1987.
- [34] R. B. Kearfott. *Rigorous global search: continuous problems*. Dodrecht: Kluwer Academic Publishers, 1996.
- [35] R. B. Kearfott, M. Dawande, K. S. Du, and C. Y. Hu. INTLIB, a portable FORTRAN 77 interval standard function library. *ACM Transaction on Mathematical Software*, 20:447–459, 1994.
- [36] R. B. Kearfott and V. Kreinovich. *Application of Interval Computations*. Kluwer Academic Publishers, Netherlands, 1996.
- [37] R. B. Kearfott and M. Novoa. INTBIS, a portable interval newton/bisection package. *ACM Transaction on Mathematical Software*, 16(2):152–157, 1990.
- [38] R. Klatte, U. Kulisch, M. Neaga, D. Ratz, and C. Ullrich. *PASCAL-XSC: A PASCAL Extension for Scientific Computation*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1991.

- [39] O. Knuppel. PROFIL/BIAS—a fast interval library. *Computing*, 53(3-4):277–287, 1994.
- [40] L. V. Kolev. *Interval methods in circuit analysis*. World Scientific, 1993.
- [41] R. Krawczyk and A. Neumaier. Interval slopes for rational functions and associated centered forms. *SIAM J. Numerical Analysis*, 22:604–616, 1985.
- [42] U. Kulisch and L. B. Rall. Numerics with automatic result verification. *Mathematics and Computers in Simulation*, 35(5):435–450, Nov. 1993.
- [43] K. N. Lalgudi, M. C. Papaefthymiou, and M. Potkonjak. Optimizing computations for effective block-processing. *ACM Transactions on Design Automation of Electronic Systems.*, 5(3):604–630, 2000.
- [44] J. D. Lawrence. *A catalog of special plane curves*. Dover, New York, 1972.
- [45] A. P. Leclerc. *Efficient and reliable global optimization*. PhD thesis, The Ohio State University, Columbus, Ohio, USA, 1992.
- [46] R. W. Maier, J. F. Brennecke, and M. A. Stadtherr. Reliable computation of azeotropes. *Computers and Chemical Engineering*, 24:1851–1858, 2000.
- [47] The MathWorks Inc., MA, USA. *MATLAB user guide, version 6.1*, 2002.
- [48] R. E. Moore. *Interval analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1966.
- [49] R. E. Moore. *Methods and applications of interval analysis*. SIAM, Philadelphia, 1979.
- [50] R. E. Moore, E. Hansen, and A. Leclerc. Rigorous methods for global optimisation. In C. A. Floudas and P. M. Pardalos, editors, *Recent Advances in Global Optimization*. Princeton University Press, 1992.
- [51] R. E. Moore and H. Ratschek. Inclusion functions and global optimization II. *Mathematical programming*, 41:341–356, 1988.
- [52] J. J. More, B. S. Garbow, and K. E. Hillstom. Testing unconstrained optimization software. *ACM Trans. Mathematical Software*, 7(1):17–41, 1981.
- [53] A. Morgan and V. Shapiro. Box bisection for solving second degree systems and the problem of clustering. *ACM Trans. Math. Software*, 13:152–167, 1987.
- [54] A. P. Morgan. *Solving polynomial systems using continuation for engineering and scientific problems*. Prentice-Hall, Englewood Cliffs, N. J., 1987.
- [55] S. G. Nash and J. Nocedal. A numerical study of the limited memory BFGS method and truncated Newton method for large scale optimization. *SIAM J. Optim.*, 1:358–372, 1991.

- [56] P. S. V. Nataraj and G. Sardar. Computation of QFT bounds for robust sensitivity and gain-phase margin specifications. *Trans. of the ASME Journal of Dynamic Systems, Measurement and Control*, 122:528–534, September 2000.
- [57] P. S. V. Nataraj and G. Sardar. Template generation for continuous transfer functions using interval analysis. *Automatica*, 36:111–119, 2000.
- [58] P. S. V. Nataraj and S. Sheela. A new subdivision strategy for range computations. *Reliable Computing*, 8:1–10, 2002.
- [59] P. S. V. Nataraj and S. Srivastava. Synthesis of robustly stabilizing general order compensators for interval plants using interval analysis. *Reliable Computing*, November 1999.
- [60] A. Neumaier. personal communication.
- [61] A. Neumaier. The enclosure of solutions of parameter dependent systems of equations. In R. E. Moore, editor, *Reliability in computing: the role of interval methods in scientific computations*. Academic Press, 1988.
- [62] A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, Cambridge, England, 1990.
- [63] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in Fortran 90 : The art of Parallel Scientific Computing*. Cambridge University Press, MA, USA, 1996.
- [64] H. Ratschek and J. Rokne. *Computer methods for the range of functions*. Chichester: Ellis Horwood Limited, 1984.
- [65] H. Ratschek and J. Rokne. *New computer methods for global optimization*. New York: Wiley, 1988.
- [66] D. Ratz and T. Csendes. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization*, 7:183–207, 1995.
- [67] J. M. Rodrigues, Y. Chait, and C. V. Hollot. An efficient algorithm for computing QFT bounds. *Trans. of the ASME Journal of Dynamic Systems, Measurement and Control*, 119(3):548–552, 1997.
- [68] S. M. Rump. INTLAB - interval laboratory. In T. Csendes, editor, *Developments in reliable computing*. Kluwer Academic Publishers, 1999.



- [69] C. A. Schnepper and M. A. Stadtherr. Robust process simulation using interval methods. *Computers and Chemical Engineering*, 20:187–199, 1996.
- [70] M. J. Schulte and E. E. Swartzlander. Variable precision, interval arithmetic coprocessors. *Reliable Computing*, 1:47–62, 1996.
- [71] M. Sidi. Feedback synthesis with plant ignorance, nonminimum phase, and time-domain tolerances. *Automatica*, 12:265–271, 1976.
- [72] M. A. Stadtherr and C. A. Schnepper. Robust phase stability analysis using interval methods. In *AIChE Symposium series*, volume 91, pages 356–359. 1995.
- [73] Sun Microsystems, Palo Alto, CA, USA. *Forte FORTRAN 95 User Manual*, 2001.
- [74] T. Sunaga. A basic theory of communication. *Memoirs*, 2,G-1:426–443, 1958.
- [75] D. F. Thomspon and O. D. I. Nwokah. Analytical loop shaping methods in quantitative feedback theory. *Trans. of the ASME Journal of Dynamic Systems, Measurement and Control*, 116:169–177, 1994.
- [76] A. Ushida and L. O. Chua. Tracing solution curves of nonlinear equations with sharp turning points. *Circuit Theory and Applications*, 12:1–21, 1984.
- [77] R. J. Vanderbei and D. F. Shanno. An interior point algorithm for nonconvex nonlinear programming. *Comp. Optim. Appl.*, 13:231–252, 1999.
- [78] J. Verschede. PHC pack: The data base of polynomial systems. Technical report, Univ.of Illinois, Mathematics Dept., Chicago, U.S.A, 2001.
- [79] E. Walter and L. Jaulin. Guaranteed characterization of stability domains via set inversion. *IEEE Trans. on Automat. Control*, 39(4):886–889, 1994.
- [80] M. Warmus. Calculus of approximations. *Bull. Acad. Pol. Sci.*, IV(5):253–259, 1956.
- [81] A. Wiethoff. *Verifizierte globale optimierung auf parallelrechnern*. PhD thesis, Universitat Karlsruhe, Karlsruhe, Germany, 1998.
- [82] M. Zettler and J. Garloff. Robustness analysis of polynomials with polynomial parameter dependency using Bernstein expansion. *IEEE Trans. on Automat. Control*, 43(3):425–431, 1998.
- [83] J. Zhou and K. A. Ross. Implementing database operations using SIMD instructions. *ACM SIGMOD June 4-6*, 2002.

## Publications

1. P. S. V. Nataraj, S. Sheela and A. K. Prakash, *Interval QFT: A Mathematical and Computational Enhancement of QFT*, Proceedings of the 5th International Symposium on Quantitative Feedback Theory and Robust Frequency Domain Methods, 23-24 August 2001, Pamplona, Spain.
2. P. S. V. Nataraj and A. K. Prakash, *A Parallelized Version of the Covering Algorithm for Solving Parameter-Dependent Systems of Nonlinear Equations*, Reliable Computing, 2002, issue 2, Volume 8, pp 1-8.
3. P. S. V. Nataraj and A. K. Prakash, *On Speeding up the Interval Newton Algorithm*, Proceedings of International Conference on Multimedia and Design, Volume II, pp 9-17, September 2002, Mumbai, India
4. M. Deo, N. S. Kubal, A. K. Prakash and P. S. V. Nataraj *Interval Algorithm to Train Artificial Neural Networks*, Proceedings of International Conference on Multimedia and Design, Volume II, pp 153-164, September 2002, Mumbai, India.
5. P.S.V. Nataraj, A.K. Prakash and N. S. Kubal, *An Efficient Algorithm of Global Optimization in Interval Analysis*, Proceeding of the 6<sup>th</sup> International Conference on High Performance Computing in Asia Pacific region, December 16-19, 2002, Bangalore, India.
6. P. S. V. Nataraj, S. Srivatava and A. K. Prakash *Design of Robustly Stabilizing Controllers for Jet Engines using Set Inversion* Proceedings of the International Symposium on Process System Engineering and Control, 3-4 January 2003, Indian Institute of Technology, Bombay, India (to appear).
7. P.S.V. Nataraj and A. K. Prakash, *On Speeding up a Basic Global Optimization Algorithm in Interval Analysis*, communicated to Reliable Computing.
8. P.S.V. Nataraj and A. K. Prakash, *A New Rule for Bisection Direction Selection in Range Computations*, communicated to Reliable Computing.

## Acknowledgement

It is a pleasant task to express my gratitude to all those near and dear ones who have accompanied and helped me in my PhD journey.

First and foremost I would like to thank my supervisor Prof. P. S. V. Nataraj, who introduced me to the wonderful world of interval mathematics. He has made a deep impression in me with his way of teaching and his simplicity. I have learned a lot from his dedication, perfection and his attitude of "never say die". He is a source of never-ending inspiration for me. I have been extremely lucky to have him as my mentor!

Gratitude is due to Prof. B. Bandopadhyay, who is also convener of our group, for spending his valuable time to read and understand my research topics. His sweet way of asking questions at the end of the seminar and 'nodding the head once convinced' will always be remembered. I am thankful to Prof. A. K. Varma who is the chairman of my RPC. I am indebted to Prof. B. V. Limaye, a wonderful teacher, who taught me Real Analysis and took special interest in my tutorials. It is befitting at this point to mention Prof. S. K. Katti for the inspiration he provided me during my masters in engineering. I am also thankful to Prof. Arnold Neumaier for his valuable comments and criticism on the second chapter of my thesis. My sincere thanks are due to Prof. S. Rump, for the software INTLAB, which I have used extensively in my work, and also for clarifying my doubts regarding the use of INTLAB with a particular version of MATLAB.

The long working hours in our lively syscon lab have been wonderful. My friends Sachin and Kubal were always there to help me out and I am definitely going to miss the coffee breaks in the coffee shack. Our discussions about anything under the sun, particularly interval mathematics will always be remembered. I am thankful to them for their support, affection and love they have shown. My friends Telang, Mariappan, Rajeev, Jignesh, Chakravarthini, Sheela, Murali and Janardhanan have been with me always sharing my experiences and providing help.

I would not have been what I am today if not for my late grandfather Mr. Sreedhara Menon - a great inspiration for me. My parents have contributed with their support throughout my PhD. My mother-in-law's presence with my children gave me the strength to concentrate on my work here. This was in spite of the health problems of my father-in-law who has always motivated me. The sacrifices they have made, staying separate, is something for which I would always be indebted to them. I am very grateful to my wife Manjula, who has been a constant source of strength. A pillar of perseverance and patience, she has been able to take care of my family along with her hectic schedule in the office beautifully. One of the best experiences that we shared in this period was the birth of our second daughter Devaki, who provides a joyful dimension to our life. My daughters Kalyani and Devaki have terribly missed their 'Acha' (papa) during the years of my research. Even with physically present with them, my

mind was most of the time preoccupied with my research problems. They have been very understanding that their papa also has to learn a lot of "a,b,c" and "1,2,3". I am proud that they have been able to take all these situations in their stride. Hopefully they will read my thesis sometime in future and ask me something about intervals!

I would also like to thank my school teachers from my village in Kerala who contributed greatly to my formative years and further development. The chain of gratitude is never-ending - there are so many people who have played significant roles in my success. Though I may not be able to mention all their names here, deep in my heart is an overwhelming feeling of warmth and gratitude for them.

Finally I am thankful to God for inspiring and guiding this humble human being, without whose blessings, nothing would have been possible!